

INGInious for Kids

Dissertation presented by
Florian THUIN

for obtaining the Master's degree in
Computer science
Option(s): Software engineering

Supervisor(s)
Olivier BONAVENTURE

Reader(s)
Anthony GEGO, Olivier GOLETTI, Chantal PONCIN, Pierre REINBOLD

Academic year 2016–2017

ABSTRACT

This master thesis covers the integration of problems solvable with Blockly blocks into INGINious. As the technology evolves and the digital world takes a greater part in our daily lives and jobs, the number of computer scientists does not follow the trend. Researchs show that kids who have a computer science course before university are more likely to enroll in a computer science degree. Nevertheless, there is still a lack of resources for teachers and kids to teach and learn the core concepts of computer science and programming.

Currently, INGINious is a platform which is used to automatically grade computer science projects at Université catholique de Louvain. This work aims to extend this platform to allow problem solving using a visual programming language in order to use it to teach programming to kids from 10 to 14 years old. The implementation has already been used to teach sequential execution, conditions, loops and functions to more than 150 french-speaking kids from Belgium.

ACKNOWLEDGMENTS

Many thanks to Olivier Goletti for his advices all over this semester to improve the usability of my implementation, his support, experience, time and advices given during the activities of Printemps des Sciences and his proofreading of my thesis.

I would like to thank Anthony Gego for the time taken during this semester to help me to setup INGIInious, his advices for the implemnetation and his proofreading of this thesis.

I would like to thank Chantal Poncin and Pierre Reinbold for their advices during the year to improve the content of the task and for their help that allowed the activities with kids to take place under very good conditions.

I would also like to thank Olivier Bonaventure, that accepted to be the supervisor of this thesis and that provided very good advices for the writing of this dissertation.

CONTENTS

I	THESIS	1
1	INTRODUCTION	3
1.1	Visual programming languages	3
1.2	INGInious	4
1.3	Problem	4
1.4	Motivation	5
1.5	Goals	5
2	OVERVIEW OF VISUAL PROGRAMMING LANGUAGES AND CHOICES	7
2.1	Blockly	9
2.2	Scratch	10
2.3	Snap!	12
2.4	Choice motivation and explanation	13
3	TOOLS DESCRIPTION	15
3.1	INGInious	15
3.1.1	History	15
3.1.2	Architecture	15
3.1.3	Interface	17
3.2	Blockly	18
3.2.1	Interface	19
3.2.2	Blocks	19
4	IMPLEMENTATION	23
4.1	Blockly integration to INGInious	23
4.1.1	Frontend	23
4.1.2	Client-side	25
4.1.3	Blockly options	28
4.2	Content creation	30
4.2.1	Definition of a new block, code generation and its interpretation	32
4.3	Blockly task creation tools	33
5	PEDAGOGICS	37
5.1	Printemps des sciences	37
5.1.1	Description	37
5.1.2	Material	37
5.1.3	Analytics	40
5.2	How to take advantage of Blockly on INGInious	48
5.3	When to avoid Blockly on INGInious?	49

5.4	Best practices	49	
5.4.1	Problem presentation		50
5.4.2	Problem content	50	
5.4.3	Test feedback	51	
5.5	Mistakes to avoid	52	
5.6	Summary	53	
6	CONCLUSION	55	
6.1	Related work	55	
6.1.1	BlockPy	55	
6.1.2	Code.org	55	
6.2	Further work	55	
6.2.1	Blockly for INGINious		56
6.2.2	INGInious	56	
6.3	Conclusion	56	
II	APPENDIX	59	
A	APPLICATIONS BASED ON BLOCKLY		61
B	RESULTS OF THE SURVEY	63	
B.1	First activity	63	
B.2	Second activity	64	
B.3	Third activity	67	
B.4	Fourth activity	69	
B.5	Fifth activity	71	
B.6	Sixth activity	74	
B.7	Eighth activity	75	
	BIBLIOGRAPHY	85	

Part I

THESIS

INTRODUCTION

Computer science education includes a lot of concepts in software and hardware. This thesis will only focus on software and particularly on providing a platform for teaching to kids the basic knowledge needed for programming with a computer. To limit the scope of this work, the choice was made to use an existing platform, INGINious, in which problems solvable using a visual programming language will be integrated.

The concept of visual programming language is introduced first (in Section 1.1), followed by a short description of the INGINious platform (in Section 1.2). This work was made to provide a platform for teaching programming to kids, the problem and the motivation behind it is described in Section 1.3–1.4 and the goals we wanted to reach are described in Section 1.5.

1.1 VISUAL PROGRAMMING LANGUAGES

A programming language is a formal language that is designed to communicate instructions to a computer. It is mostly defined by two characteristics: its syntax and its semantics. The syntax is used to say if a representation (usually text) is valid for a language, using a set of rules defined in a grammar. The semantics defines the meaning of a valid representation. When you want to write a program, it has to be syntactically correct (it has to respect the grammar of the language) to be translated into executable code, and it has to be semantically correct (act as what you want it to do).[1]

A visual programming language (VPL) has the same purpose but instead of using text, the user creates program using graphical elements. Its main purpose is to be more friendly to its users than textual programming languages, e.g. by being closer to the domain of the problems it solves or by reducing difficulties to write a syntactically correct program.

When beginners start programming, they don't find it accessible because they have to discover both the syntax and the semantics at the same time [2]. The syntax of a programming language can be hidden in pictures (i.e. for very young kids) or blocks to focus on the semantics, as shown in Figure 1.1 (the equivalent in a textual programming language is shown in Listing 1.1). A description of different visual programming languages is presented in Chapter 2.

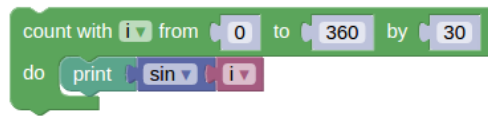


Figure 1.1: Blockly blocks

```

1 import math
2
3 i = None
4 for i in range(0, 361, 30):
5     print(math.sin(i / 180.0 * math.pi))
6

```

Listing 1.1: Python code equivalence

1.2 INGINIOUS

INGInious is an intelligent grader that allows secured and automated testing of code made by students on a web platform. It was initially developed by students and since 2014, it has become an open-source project maintained by the INGI department at Université catholique de Louvain (UCL). It is used to submit and test most of the projects in the computer science courses.[3]

INGInious includes a lot of features that are useful for computer science education. You can create user accounts, classrooms, courses, stores submissions, test input code, ask multiple choices questions, . . . This allows students to submit code from their browsers to see if it respects the instructions given by a teacher. Teachers can customize their classrooms, courses, tasks and tests to automate the grading process of student code within a safe environment. A more detailed description of INGINious can be found in Section 3.1

INGInious is extensible to new types of problems, so it can be used as a development environment for VPL problems alongside with all its previously mentioned features. The extension of INGINious with VPL problems will be described in Section 4.1.

1.3 PROBLEM

Our everyday life is being modified a lot since the last decades as the technologies evolve. In our country, every student succeeding secondary school has used a computer.[4] Across the globe, the economy tends to offer always more wages for the computer occupations.

At the same time, computer science is still less than 3% of all bachelor's degrees and only 8% of all Science, Technology, Engineering and Mathematics (STEM) degrees. Meanwhile, 71% of all STEM occupations are in computing [5]. The College Board, an American nonprofit organization formed to expand access to higher education [6], found that studying computer science in high school is highly correlated to university

enrollment [7].

One of the purpose of primary and secondary school is to help the cognitive development of their students to achieve their professional goals and to raise their awareness of the world they live in. Nonetheless, there are still very few initiatives to teach computer science to kids in Europe [8]. This problem cannot be solved if teachers in Belgium do not have an education platform where they can make (or use existing) exercises that are available to their students in French.

1.4 MOTIVATION

Every active person uses softwares on a daily basis (at home, at work or even to buy a public transport ticket), but very few understand how it is made. One of the goal of education is to help people understand the world in which they live, so understanding how programs are made is an interesting topic that could be added in today's school program.

The usage of a **VPL** allows to create syntactically correct programs, but it also takes the advantages of the colors and shapes, which are more expressive than indentation and coloring to make the source code more readable. Moreover, with a VPL the functions available are shown to the user such that he doesn't have to read documentation, which is also a nice advantage for beginners.

Thus, using a **VPL** is probably one of the most practical way to start learning core programming concepts as a kid. Integrating it into an educational platform that already provides an environment to display exercises, test submissions and has a community of active users and developers allows to save more time to develop content and test it with some classes, as it will be described in Section 5.1.

1.5 GOALS

This master thesis has four main objectives:

1. **Choosing a visual language:** There exists different visual language with their advantages and drawbacks, the goal is to meet the needs of kids but also to integrate it in INGINious. This choice will be explained in Section 2.4;
2. **Integration to INGINious:** The goal is to use the advantages of INGINious: submissions testing, storage, classrooms,... Integrating a new problem is not something that is usual, and INGINious does not handle anything that is visually

interactive with the code (only text boxes or files). The integration is explained in Section 4.1;

3. **Content creation:** The goal is to show that the integration is a success by creating different exercises that corresponds to interesting real-life activities solvable by kids. The content creation process is described in Section 4.2 and more created content is described in Section 5.1.2;
4. **Real-life test:** The goal is to prove that the content and the platform allows to organize an activity with a class of kids (10-15 years) to help them to learn some core programming concepts. During a week, eight activities were organized, each time with different classes, the results of those activities can be found in Section 5.1.3. Section 5.4 will describe some good practices learned from those activities.

OVERVIEW OF VISUAL PROGRAMMING LANGUAGES AND CHOICES

Since the 1990s, a lot of work has been made to design and implement visual programming languages to help people without any knowledge in computer science to learn the core concepts of coding. In this chapter, different visual programming languages will be shown but most of them do not fit the requirements leading us to my choice in Section 2.4.

The requirements are:

- A french translation is available or possible to do;
- It can be integrated inside INGINious with little to no modifications of the existing implementation. It is important to avoid breaking or hacking the existing version of INGINious and to be able to take advantages of the existing features of INGINious;
- There is an active community of developers and users;
- It looks friendly for kids (colors, animations, sounds,...);
- It supports multiple paradigms.

The oldest programming languages used all over the world that survived until today are AgentSheets[9] and Etoys[10]. AgentSheets is developed in Java, it provides an easy interface to build 2D and 3D games and simulation for kids and scientists (as shown on Figure 2.1). Etoys is an application with a drawing interface where the drawings can be programmed to move to learn sequencing, conditional statements, loops and object-oriented programming (as shown on Figure 2.2). Those applications developed since 1996 are available as desktop applications for Windows and MacOS.

Another kind of visual language projects is plugins for existing Integrated Development Environment (IDE). An example is the Ardublock project that integrates in the ArduinoIDE to allow kids to program their Arduino hardware with blocks, as shown in Figure 2.3. The main difference between Ardublock and Etoys or AgentSheets is that the visual language (OpenBlocks) is used to be translated into a real-world language to be compiled and run, so it can help to introduce or to ease the use of a textual language.

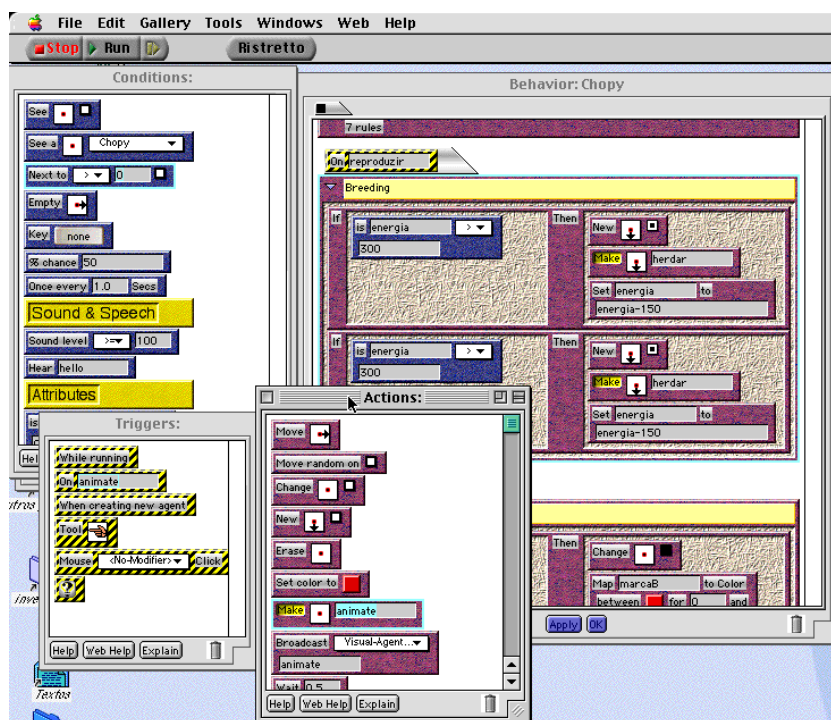


Figure 2.1: Agentsheets interface

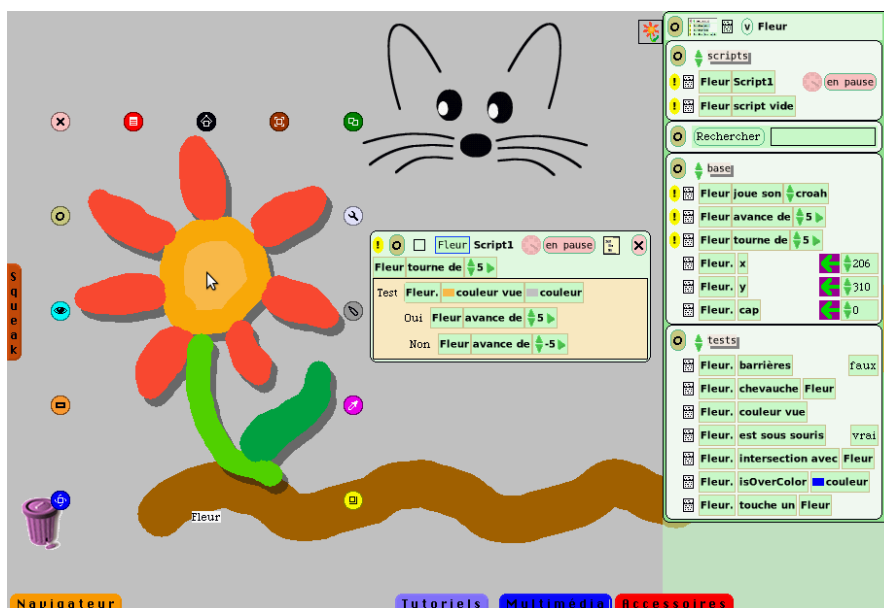


Figure 2.2: Etoys interface

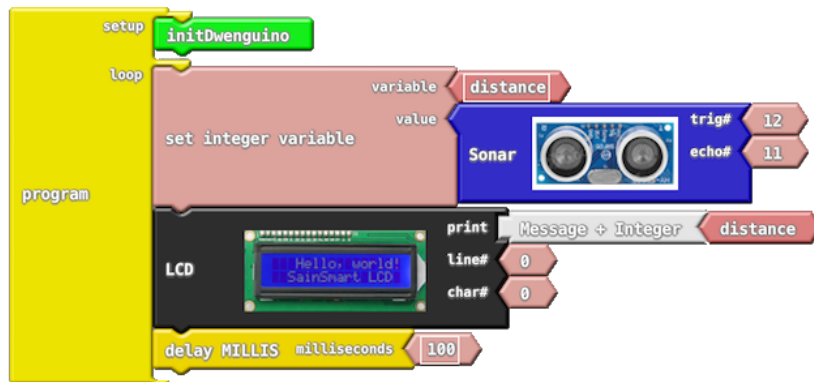


Figure 2.3: Ardublock interface

Nowadays, the trend is to get rid of any installation process and operating system requirement and to be able to create and execute programs using **VPL** in a web browser. As for desktop applications, there are two main kinds of way it can work: whether the visual language is to be used on its own, or the visual language is translated to textual code. The first kind is illustrated in projects such as Scratch or Snap!, their applications is presented as a full page where the only way to build your program is to use the given blocks to animate and move elements on a sprite. The latter kind is illustrated by Blockly where the blocks are only a handy representation of a portion of text in a real-world language. As those web-based **VPL** are the ones which could fit for an integration in INGIInious with the biggest and more active communities, the following sections will describe them more deeply.

2.1 BLOCKLY

Blockly is a visual language that is developed by Google since 2013. The idea was to replace OpenBlocks which was developed in Java to allow an integration in a web browser [11]. Blockly is fully written in JavaScript to allow its integration in all modern browsers (there are also versions for iOS and Android). The Blockly project defines several blocks for core concepts, such as conditions, loops, functions, variables, colouring, math functions, ... as shown on Figure 2.4.

Blockly is translated in more than 40 languages including French. The interface is made to be included in any web page and works without a server. The community of developers is active and the community of projects using Blockly (e.g. Code.org) is large and very active. The interface is rather simple to use because it only consists in click and drag-and-drop without any buttons by default. The Blockly blocks library is made to be extended by new blocks related to each exercise, so it is not restricted to

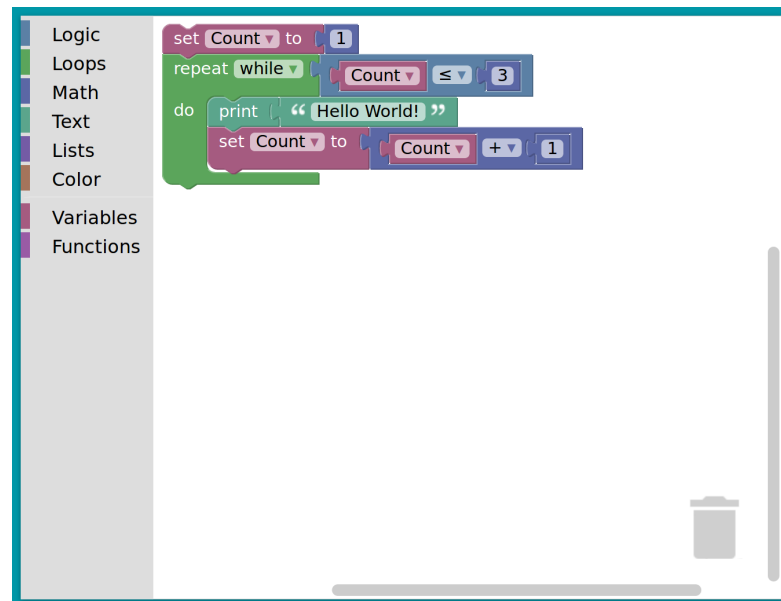


Figure 2.4: Blockly interface as shown on the official website

any paradigm.

With Blockly, all blocks have to be translated in an existing textual language. The blocks already defined are translated in 5 languages: JavaScript, Python, PHP, Dart and Lua. As the translation is syntactically correct in any use of each block, it ensures that any program created with blocks is syntactically correct. This requires that the programmer pays a lot of attention when he creates a new block to test all its possible use but it also gives the opportunity to easily create new blocks by only defining a shape and a translation.

2.2 SCRATCH

Scratch is an educational project developed by the MIT since 2006. The first version of Scratch was a desktop application written in Smalltalk. It switched to ActionScript (Flash) for its second version, available at <https://scratch.mit.edu> (interface is shown in Figure 2.5).

Since its launch, Scratch has registered more than 19 millions of users who have shared 23 million of projects [12], making it the biggest community for a visual programming language. It is translated in many languages including French. The ease of integrating Scratch into an existing platform is arguable: Scratch is a Flash application

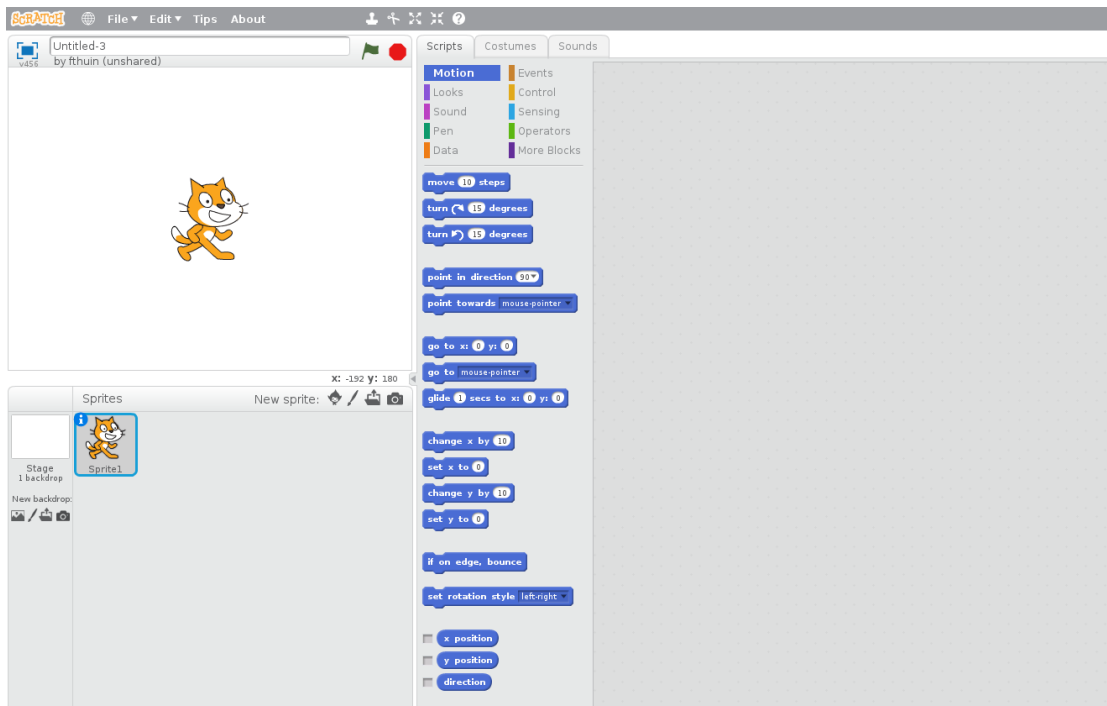


Figure 2.5: Scratch interface

so technically it is just an element that can be integrated in any page, but at the same time, Scratch is not made to reach a predefined goal (the goal is to imagine something, e.g. a game, and then program it). This means that the program build with Scratch are not made to be testable but to be shareable (every project can become a standalone application). The work to make Scratch take use of INGINIOUS would be large: first you have to provide a way to compile and test programs created with the editor, and then you would have to make the editor communicate with the existing platform, knowing that it was not made for those purposes.

Moreover, the Scratch team is now working on a third version of the platform, they want to get rid of Flash (because the most used browsers are disabling Flash player by default [13][14], mainly for performance or security reasons) by switching to JavaScript. They divided the new version into three projects:

1. **Scratch blocks** is a fork of Blockly project, extended with blocks for interaction with the Scratch GUI.
2. **Scratch VM** is a virtual machine used to represent, run and maintain the state of programs for Scratch 3.0. Its goal is to provide an execution environment (a JavaScript interpreter) together with abstract syntax tree of the code generated by blocks.

3. **Scratch GUI** is a graphical user interface developed using React framework to view, modify and run Scratch projects.

Concretely, Scratch blocks will share the same basis as Blockly in its next version. Furthermore, the next version of Scratch will probably remove the difficulty of integration mentioned above, because the VM will expose the state of the programs (allowing testing) and the GUI will be composed of multiple components instead of being considered by the browser as a single component. Unfortunately, this version is not yet released (only a developer preview is available), so it is not possible yet to work with a fully functional and stable release.

2.3 SNAP!

Until its version 3.0, the Snap! project was called BYOB (Build Your Own Blocks), it was a desktop application running on Windows, Mac and Linux. In its 4.0 version, it has completely switched to JavaScript using the `morphic.js` framework with an online version available at <https://snap.berkeley.edu> (the interface is shown in Figure 2.6).

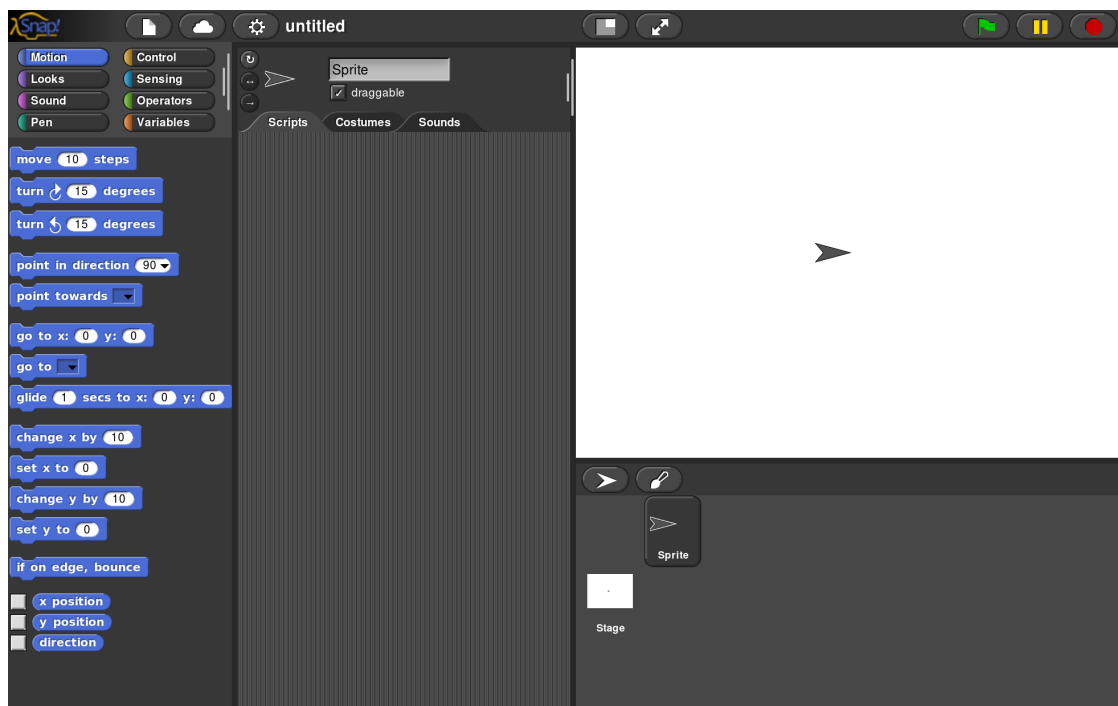


Figure 2.6: Snap interface

The major advantage of Snap! over Scratch is that it includes procedures, lists and objects as first-class citizens, which was not supported by Scratch (so it aims to handle more paradigms). It has a french translation and an active community (smaller than the Scratch one). For its integration, it shares the same drawbacks as Scratch because programs are not made to be automatically testable outside of the Snap! application and there is no communication library between Snap! interface and the platform it runs on (apart from optional features such as login).

2.4 CHOICE MOTIVATION AND EXPLANATION

As presented in the previous section, there are only three projects with big communities that can be used inside a web-based platform. One of the point that was raised in the introduction was the lack of resources to teach programming, and if you don't have a large community you will not be able to find help easily, that's why it is an important criterion.

Another important criterion to take into account how and why INGINious is made (the technical details will be more deeply discussed in Section 3.1). As explained in Section 2.2 and 2.3, programs created using Scratch and Snap! are made to be tested from outside of their respective applications and their interface is not made to communicate with external platform. Concretely, it would require a lot of work to modify their existing behavior to fit with the existing test platform of INGINious.

An example of the way you can integrate Snap! and test programs created by its application was done in two separate master thesis in 2014 and 2015, the platform with Snap! was developed by Simon Claessens and Gaëtan Collart [15] and the blocks for introspection to allow the testing and some other extensions were created by Simon Hock [16]. To create tests on Snap! exercises, he added blocks to the beginning and the end of each student solution to launch the execution of tests. At the end of the execution, the interface sends to the server if the task is succeeded or failed. This kind of behavior is the opposite of what we want to reach in this master thesis: we want to use the server (INGInious) to give us the information about the success or the failure of an exercise, but this is not possible without modifying the way Snap! works.

At the other side, Blockly is made to translate the blocks into programs written in existing programming languages that are already tested with INGINious. The Blockly interface and blocks are made to communicate with the external platform it is integrated in. From an architectural point of view, it is the one that corresponds the most to the existing problems handled by INGINious and that takes most profit of what IN-

GINious can already offer. Thus, Blockly is the **VPL** that I will integrate to INGINious. The implementation will be explained in Section **4.1**.

TOOLS DESCRIPTION

3.1 INGINIOUS

3.1.1 *History*

INGInious was made by the INGI department at the Université catholique de Louvain. The two main developers are Guillaume Derval and Anthony Gego, they started the development as students during the summer holidays of 2014 to build an alternative to Pythia.

It is widely used in the computer science courses in the INGI department, more and more projects and homeworks are submitted and automatically tested on it, from the first year of bachelor to the last year of master. INGINious is still under development to improve its existing features. One of the new goal of the platform is to become a testing platform for companies that hire developers and software engineers. INGINious is mostly used with its webapp (web application), which will be described in the next section but tasks are also integratable in other learning platforms supporting LTI.

3.1.2 *Architecture*

INGInious is an open-source project developed in Python with the framework `web.py`, with Bootstrap as HTML framework and the jQuery library. It has a model-view-controller (MVC) architecture and it uses Docker containers to securely run the code produced by students.

The broad outlines of INGINious architecture is shown in Figure 3.1. Users can request web pages to the frontend, that will response with the information stored in a MongoDB database and in the files related to the courses. From specific web pages, users can submit solutions for a task to the frontend, that will interact with its back-end modules to ask to an INGINious agent to test the submission of the students. The INGINious agent creates a Docker container to launch a secure execution of given tests on the submissions (the code in a submission is ran in isolation from the rest of the server and is restricted to a certain usage following the container configuration). Once the working process of the container has terminated, it returns its results to the agent,

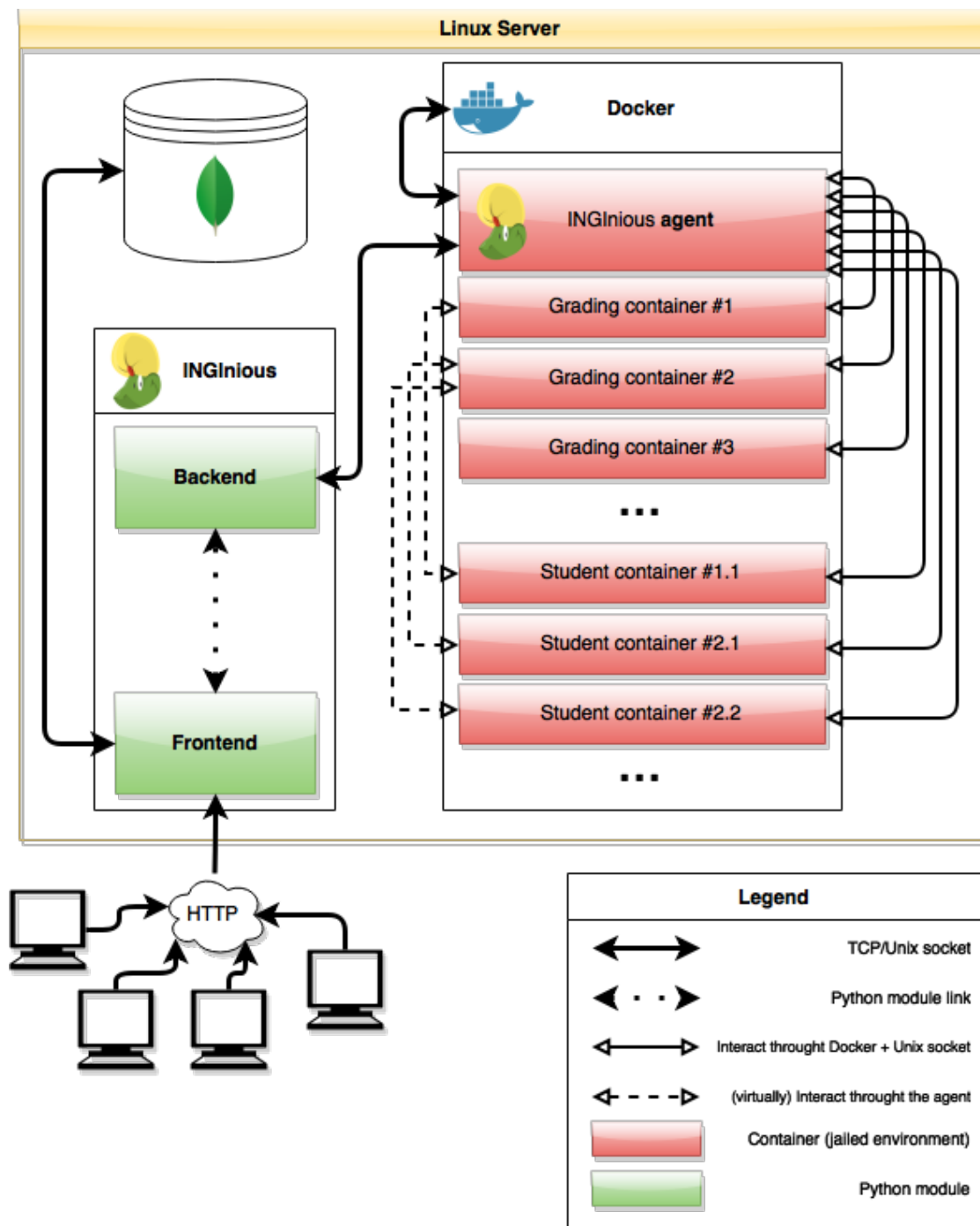


Figure 3.1: Basic architecture of INGINious

that communicates it to the backend which transmits it to the frontend that stores the results in the database and then display the related information the user.

3.1.3 Interface

The web application of INGINious when an user is logged in displays the list of courses the user is registered to, and a menu to register to new courses. When a course is clicked in the list, a page shows all tasks available for this course and their respective grades obtained by the user. Each course has to be in a different directory that contains its properties in a `course.yaml` file and this directory can contain multiple task directories, each with their properties and problems defined in a `task.yaml` file, as depicted in Figure 3.2. Each task contains one to many problems of different types:

- Single line code
- Multiple lines code
- Multiple choice question
- File upload box
- Perfect text match

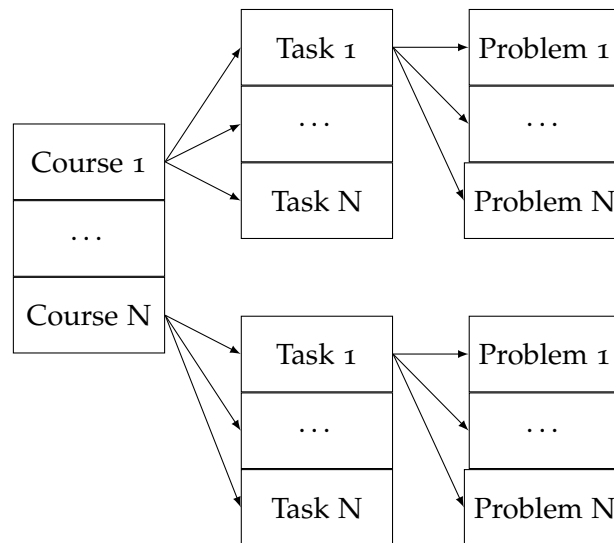


Figure 3.2: Architecture of the courses in INGINious

To complete a task, all subproblems have to be solved (empty problems are not allowed by default) to submit a task, but the grading allows to know which subproblems has been succeeded and which are not. Submitting a task will create a docker container in which the tests made by the course administrator will run. Each user can access to his older submissions on the related task page.

Course administrators can customize each task to limit the number of submissions allowed, the number of submissions stored for each user, the time and memory usage limit for the tests, the weight of the grade w.r.t other tasks, to allow or deny the internet access of the grading container.

3.2 BLOCKLY

Blockly is an extensible visual block programming language developed by Google. This open-source project was started to allow the development of Android applications through App Inventor possible in a web browser (as a replacement for the Java project OpenBlocks for desktop applications). The two main developers are Neil Fraser and Rachel Fenichel.

It is developed in pure JavaScript and is 100% client-side, with a dependency to the Closure Compiler to build the project and run some demos. The closure compiler is not a compiler, but a minifier for JavaScript code which, besides reducing the size of the JavaScript files (to reduce the web page loading time), warns about bad practices and syntax errors.

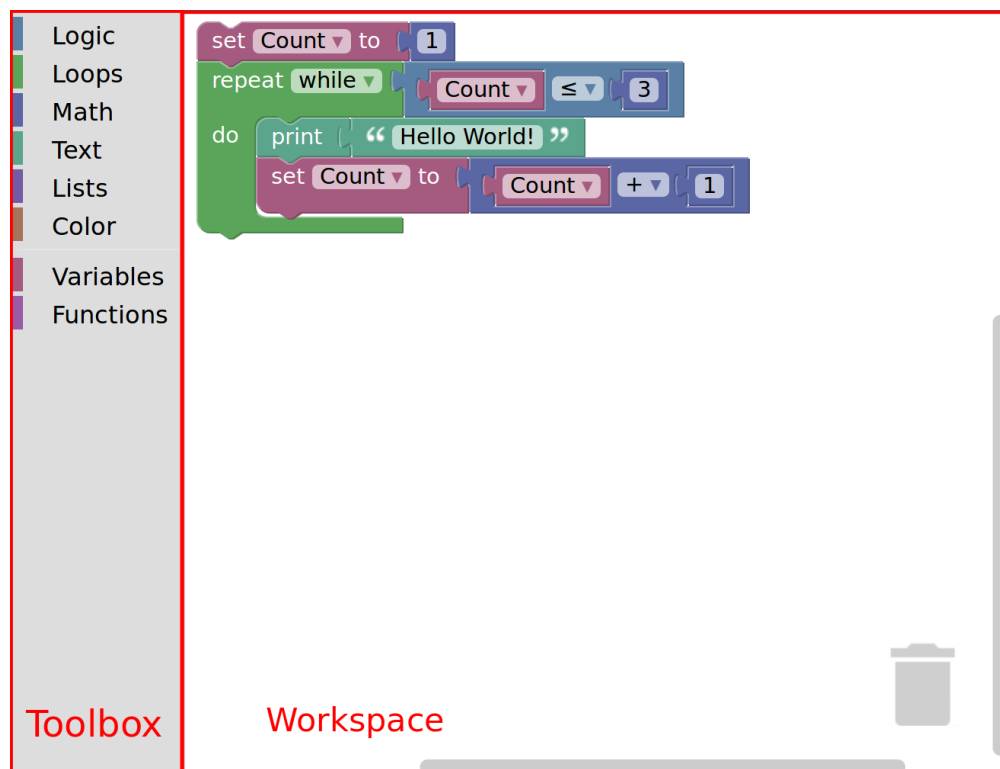


Figure 3.3: Blockly interface

3.2.1 Interface

The Blockly interface is composed of the main parts: the toolbox and the workspace, as shown in Figure 3.3. The toolbox is displayed on a border of the workspace, usually vertically on the left. It contains all the blocks available to create the program and displays them whether directly or by categories. The workspace is the place where the blocks can be stacked together to create a program.

3.2.2 Blocks

Blocks are defined in JavaScript or JSON and their translations can be done in any language, but the basic blocks are defined in 5 languages (Python, JavaScript, Lua, PHP, Dart), so if you want to use them in another language you will have to translate them.

Blockly has been made to be extended by context-specific blocks, usually blocks that are very closely related to the problem to solve. Each block has a definition to define how the block must be displayed and how it can interact with other blocks in the workspace. Each block has also a code generation part to be able to transform it into a textual language.

```

1 {
2   "type": "controls_repeat_ext",
3   "message0": "repeat %1 times",
4   "argso": [{
5     "type": "input_value",
6     "name": "TIMES",
7     "check": "Number"
8   }],
9   "message1": "do %1",
10  "args1": [{
11    "type": "input_statement",
12    "name": "DO"
13  }],
14  "previousStatement": null,
15  "nextStatement": null,
16  ...
17 }
```

Listing 3.1: Repeat block definition (JSON)

```

1 Blockly.Blocks['text_print'] = {
2   init: function() {
3     this.jsonInit({
4       "message0": "print",
5       "argso": [
6         {
7           "type": "input_value",
8           "name": "TEXT"
9         }
10      ],
11      "previousStatement": null,
12      "nextStatement": null,
13      ...
14    });
15  }
16 };
```

Listing 3.2: Print block definition (JavaScript)

3.2.2.1 Block definition

To create a new block, you have to define how it can be connected to other blocks, if it has input or output, its color, the text to be displayed inside it and a unique id. This

definition can be done in JSON or JavaScript (as shown respectively in Listing 3.1 and 3.2, with the resulting blocks in Figure 3.4 and 3.5), but developers created the Blockly Developer Tools [17] to allow to do it graphically, using blocks (those blocks have a translation to produce the definition of new blocks in JSON and JavaScript).



Figure 3.4: Blockly repeat block



Figure 3.5: Blockly print block

3.2.2.2 Block code generation

Once all the inputs and outputs have been defined in the block definition, the code translation of the block can be written in the language of your choice. This part is difficult because the translation has to be always syntactically correct, so it requires a good knowledge of the language used for the translation. An example of the blocks “repeat” and “print” is shown in Listing 3.3 and 3.4.

```

1 Blockly.Python['controls_repeat'] = function(block) {
2   var repeats = String(parseInt(block.getFieldValue('TIMES'), 10));
3   if (Blockly.isNumber(repeats)) {
4     repeats = parseInt(repeats, 10);
5   } else {
6     repeats = 'int(' + repeats + ')';
7   }
8   var branch = Blockly.Python.statementToCode(block, 'DO');
9   branch = Blockly.Python.addLoopTrap(branch, block.id);
10  var loopVar = Blockly.Python.variableDB_.getDistinctName(
11    'count', Blockly.Variables.NAME_TYPE);
12  var code = 'for ' + loopVar + ' in range(' + repeats + '):\n' + branch;
13  return code;
14 };

```

Listing 3.3: Block repeat code generation for Python

```

1 Blockly.Python['text_print'] = function(block) {
2   var msg = Blockly.Python.valueToCode(block, 'TEXT',
3     Blockly.Python.ORDER_NONE) || '\'\';
4   return 'print(' + msg + ')\n';
5 };

```

Listing 3.4: Block print code generation for Python

Those definitions and code generation implementations allow to connect the two blocks together (as shown in Figure 3.6) to generate a little program, as shown in Listing 3.5.

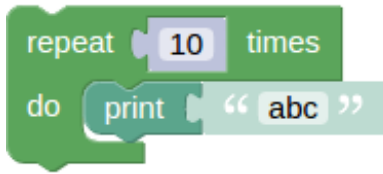


Figure 3.6: Block repeat and print used together

```
1 for count in range(10):  
2     print('abc')  
3
```

Listing 3.5: Translation of the a print block inside of a repeat block in Python

IMPLEMENTATION

4.1 BLOCKLY INTEGRATION TO INGINIOUS

This section contains a description of the modification made to the frontend of INGINious (in Section 4.1.1), the code added to the client-side in Section 4.1.2 and the options related to a Blockly problem are described in Section 4.1.3.

4.1.1 *Frontend*

As there exists no tutorial for that, the steps to add a new type of problem are shortly explained in the next paragraphs and in UML diagrams. To define a task, INGINious needs a file (always called `task.yaml`) that contains all information about the task. The starting point of the implementation is to add a new type of problem to INGINious, to allow INGINious to correctly loads that file (existing type of problems can be found in Section 3.1.3). This new type of problem is called `BlocklyProblem` and inherits from the existing `CodeProblem` which defines attributes and methods for a problem that handles code, the resulting structure is shown in Figure 4.1. A new type of problem is needed because of the specific options related to Blockly problems explained in Section 4.1.3.

`BlocklyProblem` instances contain a list of boxes created with options specific to the problem. A box contains the information that will then be used to correctly display a problem inside a web page. There are information specific to Blockly, so we need to create a new class `BlocklyBox` that inherits from the `MultilineBox` (because blocks generate multiple line of code and this is what `MultilineBox` already handles) as shown in Figure 4.2. Those information are the toolbox (a textual representation of them in a language called XML), the workspace (to define blocks that should appear in the workspace at its initialization), options to customize the workspace (zoom, grid,...) and optionally files related to new blocks definition and other scripts. An example of usage is given in Section 4.2.

To create a web page, the INGINious frontend uses a renderer that puts together partial HTML template (i.e. pieces of a web page) in which data can be added programmatically. To display a task page, the renderer calls the method `show` on each displayable box that are related to the problems in that task. So we need to define a new type of displayable box that will inherit from `BlocklyBox` (explained earlier)

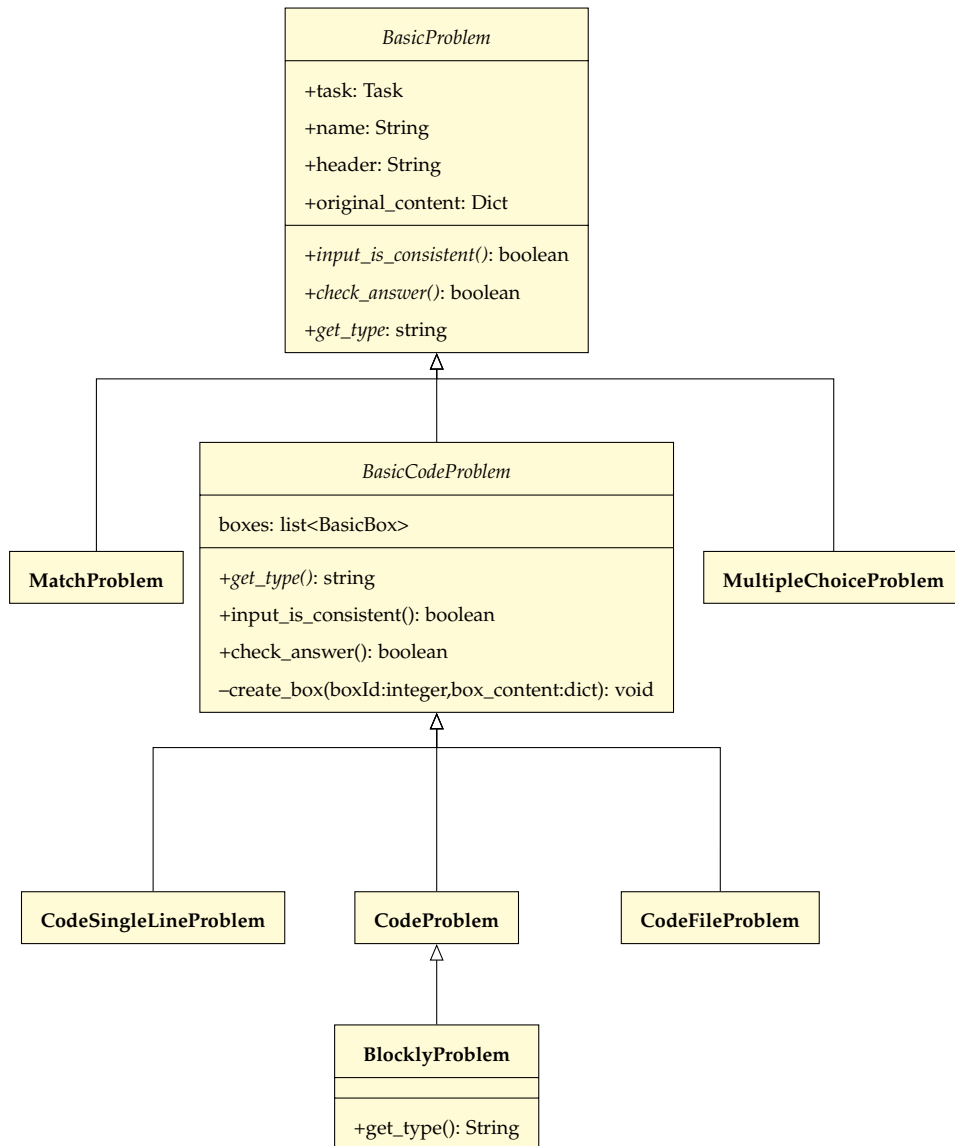


Figure 4.1: UML diagram of the Problem classes

and from `DisplayableBox` abstract class to tell the renderer how to display the parts related to a `Blockly` problem inside the task web page. This new type of displayable box is called `DisplayableBlocklyBox`, its relation with the existing implementation is depicted in Figure 4.3.

The renderer comes from the `web.py` framework used by INGINious and is not directly related to this work, so it is not explained here. Shortly, `DisplayableBlocklyBox` uses it to generate a part of a web page (as HTML text) from a template (in

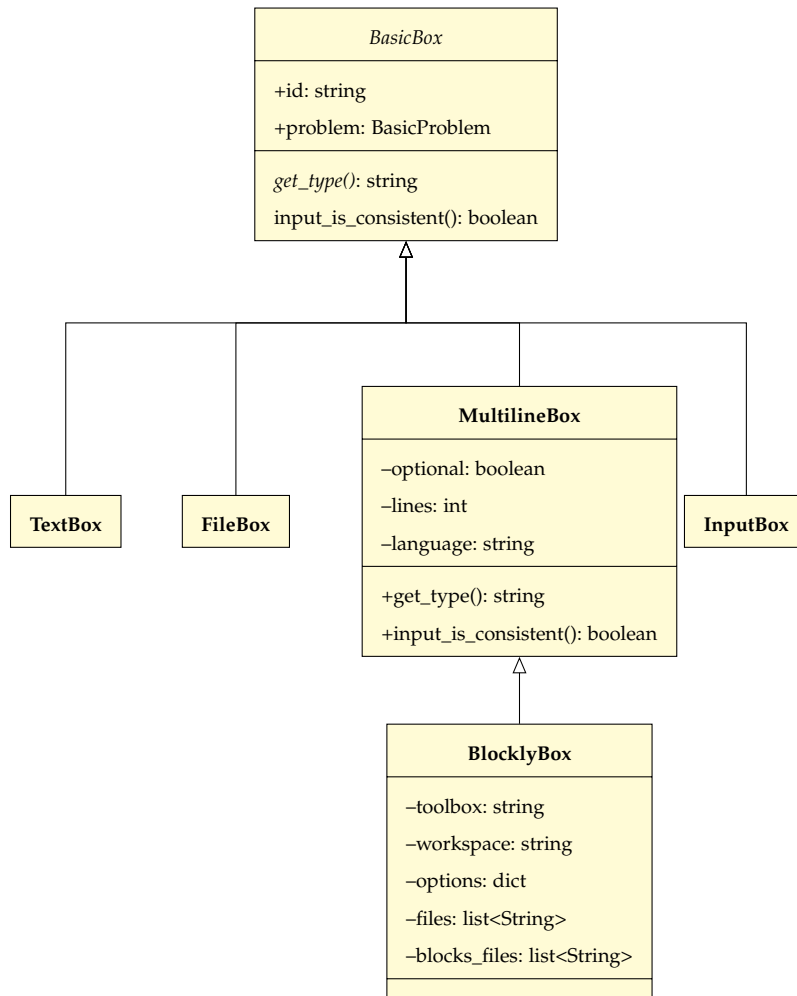


Figure 4.2: UML diagram of the Box classes

inginius/frontend/common/templates/tasks/blockly.html). The template consists mainly of HTML elements and scripts to load, so its content is not presented here.

4.1.2 Client-side

When an user loads a task with a Blockly problem, his browser will show the web page generated by the renderer and runs scripts in JavaScript to display the Blockly workspace, it has to be injected in JavaScript using the blockly library. All JavaScript files needed to display and interact with a blockly problem is in the directory `inginius/frontend/common/static/js/blockly/`.

This directory contains multiple subdirectories and files:

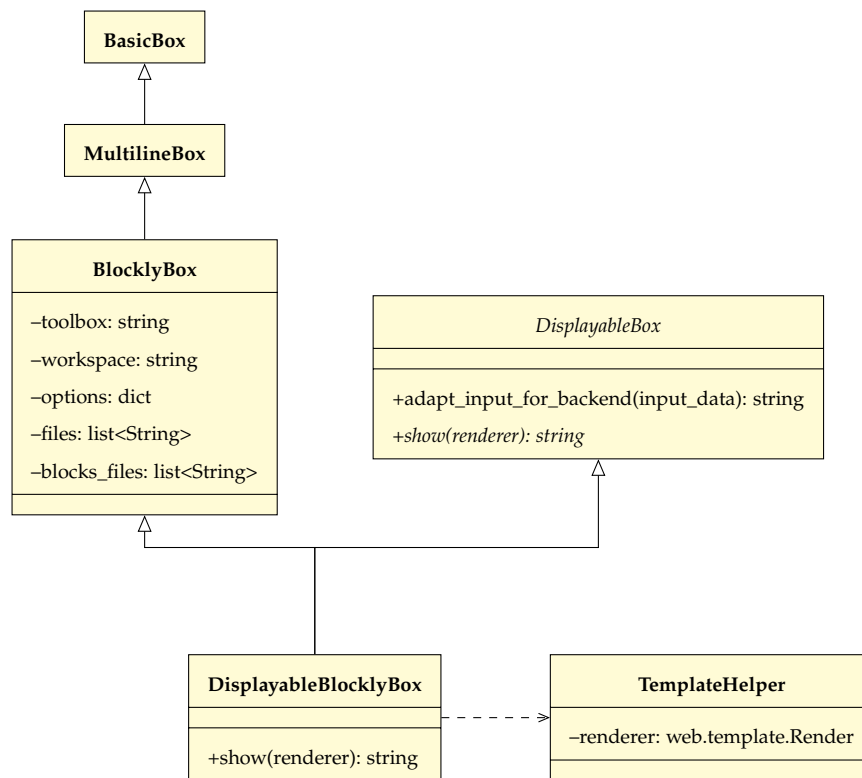


Figure 4.3: UML diagram for DisplayableBlocklyBox class

- `media/` is a directory where pictures and sounds related to Blockly are stored (sounds are played when the user moves the blocks, deletes them and pictures are needed for the trashcan and the zoom buttons)
- `msg/` is a directory containing all the translations (in more than 60 languages) of the Blockly interface and blocks. At the moment, localization is not yet supported by INGINIOUS so it has to be set manually.
- `blockly_compressed.js` contains all the functions to display and interact with the Blockly environment.
- `blockly.js` defines two classes, `BlocklyTask` and `BlocklyTaskInterpreter`. A `BlocklyTask` instance is created for each Blockly problem and it initializes a new `BlocklyTaskInterpreter` to be able to run the code locally.
- `blocks_compressed.js` contains all the core blocks available in Blockly.
- `dart_compressed.js`, `javascript_compressed.js`, `lua_compressed.js`, `php_compressed.js` and `python_compressed.js` contain the translation of the blocks into the language their names suggest.

The implementation consists mainly in an implementation of a `BlocklyTask` class in JavaScript that will handle the information and the visualization of the Blockly problem. It contains a `BlocklyTaskInterpreter` instance that is a JavaScript interpreter to interpret JavaScript code in isolation (this interpreter comes from a project developed by a developer of Blockly [18]).

The `BlocklyTask` instance related to a Blockly problem will handle its display but also the interaction with the Blockly library to keep a real-time translation of the blocks in the workspace and their representations at any moment. This real-time translation and blocks structure is stored inside hidden page elements. Such that, at any moment if the user wants to submit, the content of those elements is sent to the server (that will test the code and store the blocks structure to allow to display it if the user wants to see again his submission in the future). The content of the element storing the code generation can also be read by the `BlocklyTaskInterpreter` instance if the user ask to test his code locally.

An interpreter was needed because for kids, it is more stimulating to have a visual counterpart of their code as easily as possible. An interpreter is the most straightforward way to provide in the browser a direct execution of the code generated by blocks in order to show the result of execution (by highlighting each block and by showing their visual counterpart at the same time).

With the implementation of the frontend and the code added for the client side, a task can be displayed on any task page (an example is shown in Figure 4.4). As you can see, from top to bottom there is:

- **The problem name** to identify the problem;
- **The problem description** to display instructions to solve the problem;
- **The interpreter buttons** (alternately run code, stop, restart) to start the execution of the code inside the browser, stop it and restart from the beginning to prepare for a new execution;
- **Display modes** to show blocks, blocks and the code generated from them or only the code generated from them
- **Full screen mode** to see the problem as a full screen page;
- **Block limitation information** (optional) shows the maximum number of blocks that can be used to solve the problem.
- **Toolbox** and **Workspace** form the Blockly interface, already described in Section 3.2.

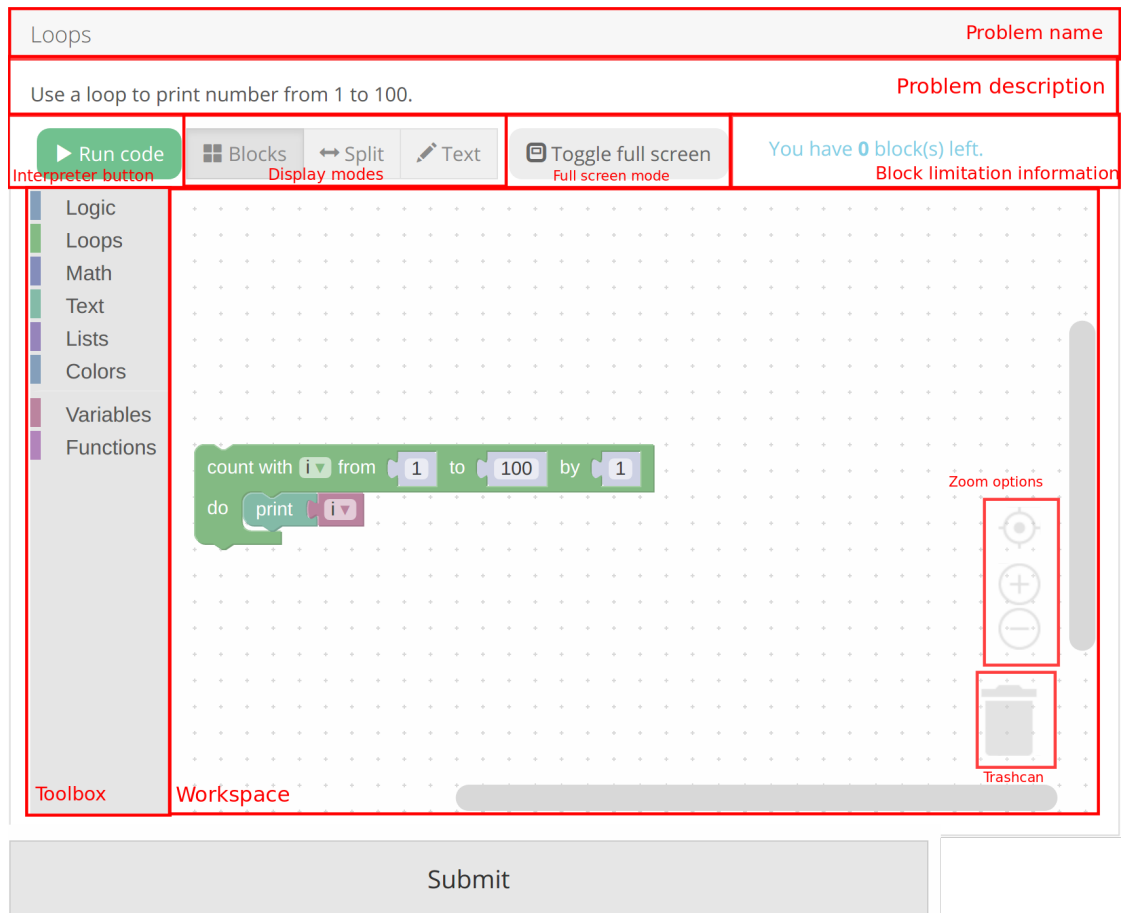


Figure 4.4: An example of Blockly problem displayed in an INGINIOUS task page

- **Submit button** to send a solution of the problem to INGINIOUS to know if the task is succeeded.

4.1.3 Blockly options

The implementation supports the customization of the Blockly workspace with a lot of different options (that can be adjusted by the teacher in the INGINIOUS task editor at any moment). Those options are:

- **Collapsible blocks** allows the user to collapse all connected blocks below a selected block (e.g. to save space on the workspace)
- **Commentable blocks** allows the user to add comment on blocks
- **Disable blocks** allows the user to disable blocks such that they do not generate code but they are not deleted (e.g. for testing purposes)

- **Maximum number of blocks in the workspace** allows the creator of the task to limit the number of blocks draggable in the workspace (e.g. to force the use of loops)
- **Trashcan** allows the user to delete its blocks with drag-and-drop
- **Horizontal layout** displays the toolbox horizontally instead of vertically
- **Toolbox position** start or end (if toolbox layout is vertical, start means left, end means right)
- **Media path** tells Blockly where to find the media resources (sounds and pictures)
- **Display grid**: display a grid on the workspace (with spacing, length, color and snap options)
- **Zoom** allows to zoom in/out the workspace with controls and/or the mouse wheel (with controls, wheel, scale options)
- **Right-to-left** allows to handle languages that you can't read from left to right.
- **Read only**: unmodifiable workspace (for example for an exercise "what would this code prints?")
- **1-based-index** allows to begin all indices by 1 instead of 0 (for lists, strings, ...)
- **Scrollbars** allows to display scrollbars to navigate in workspace
- **Sounds** activate the sounds (e.g. when blocks are stacked)

To ease the creation of tasks in INGINious, the following options were added:

- **Visual**: display a SVG zone to allow to programmatically display visual content (optional);
- **Files**: a list of files needed for the interpreter to handle the code generated by the new blocks (optional);
- **Blocks files**: a list of files containing the new blocks definition related to the task (optional);
- **Toolbox**: the list of blocks (formatted in XML) that will be displayed in the toolbox for the problem;
- **Workspace**: the list of the blocks (formatted in XML) that should be displayed in the workspace when the task is first shown (optional).

4.2 CONTENT CREATION

To test my implementation of Blockly on INGINious, I created two types of content. The first one was non-visual problems solvable without adding any new blocks based on the problems proposed in the course “Learn Python” which is an introduction to the Python language for computer scientist at UCL (available on the instance for UCL students [19]). Implementing this kind of problem is very easy to do because Blockly blocks can be translated to Python so the exact same tests as for a problem resolved with text can be applied to the generated code. An example of such task is shown on Figure 4.5, if you want to understand the shown solution, keep in mind that with Blockly by default the indexing starts to 1.

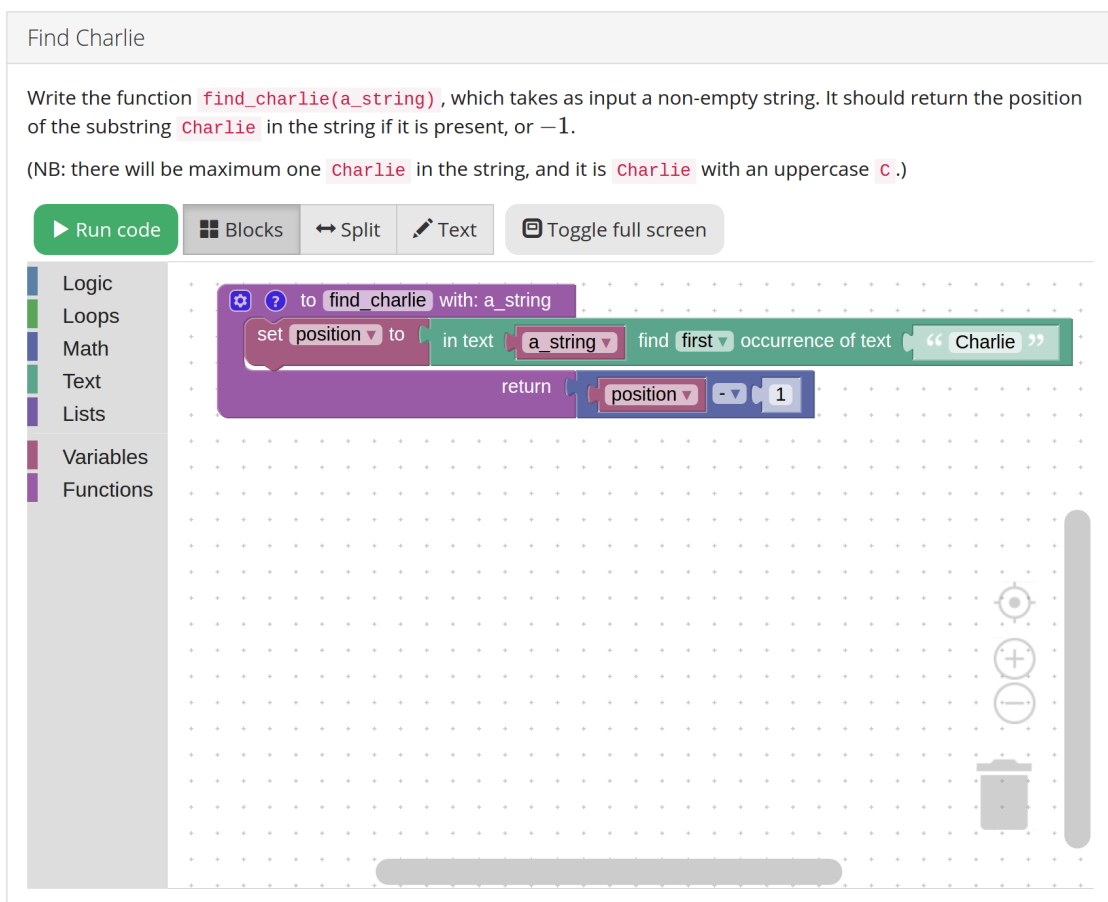


Figure 4.5: An introduction task from the course Learn Python resolved with Blockly

Those kinds of tasks that are not rich in visual content are not enjoyable the one preferred by kids [20] so I wanted to add something more visual to show how it can

be integrated into INGINious. I took an example from the Blockly Games [21] where the Maze shown in Figure 4.6 is proposed. Blockly Games are 100% client-side little games which make them rather easy to import inside another website. The advantage is that most of the code is reusable as is, but there are modifications that will be described here to show how to create content for INGINious.

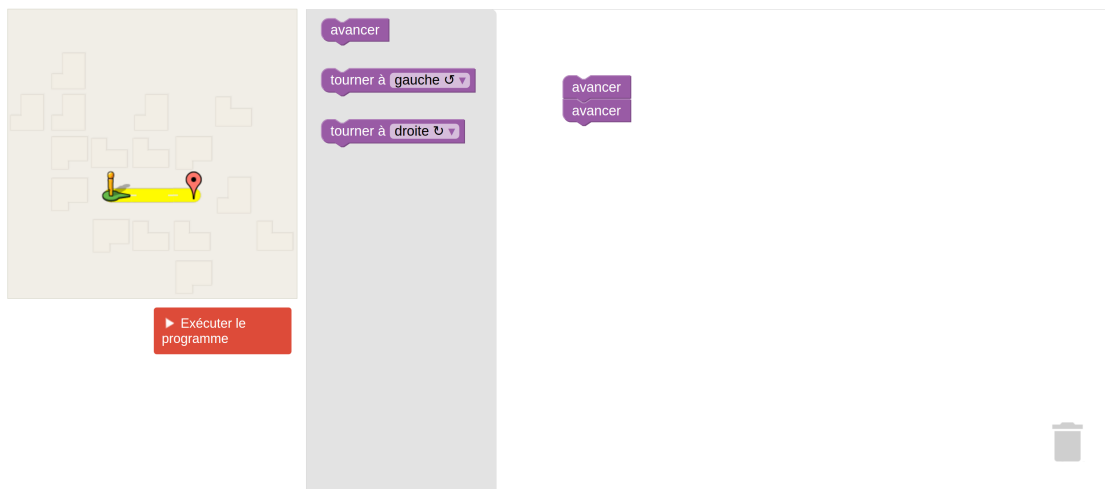


Figure 4.6: The maze activity from the Blockly Games

The first modification is related to the way levels are managed in the maze. The maze game is divided in 10 levels, and all the levels share the same files and there is only an argument (named “level”) in the URL that sets it when you click on another level. As explained in the description of INGINious (Section 3.1.3), tasks are separated in different directories (one for each task), making them independent from each other. Thus, we have to create a task for each level (because we want the students to be able to submit the level one by one). It means that the files have to be replicated in each task and parts not related to a given level can be removed.

The second modification is related to the way the interpreter works with the Maze. The interpreter integrated to Blockly for INGINious has been made to be as general as possible and as any program, it does not solve the halting problem so it might interpret an infinite loop. Unfortunately, the maze as it is implemented does not respect this idea. What is done when a user executes his program is a logging a maximum of 10,000 steps of execution and it halts if there is more. This technique is used to display the interpretation of logged steps in various speeds according to the result (the animation runs quicker if the problem is solved, slower if it is failed). This was a huge assumption in the initial code and it had to be changed to work with the more general interpreter included in INGINious (that will execute the code step-by-step until it

reaches the end of the program or is stopped by the user).

Once it has been done, the rest of the code could be reused. Next section presents an example for the definition of a new block and its code generation into Python and JavaScript language. The code generated will be a function that is related to the logic of the task and the way it can be managed by the JavaScript interpreter is shown.

4.2.1 Definition of a new block, code generation and its interpretation

First, the blocks have to be defined as shown in Listing 4.1. The definition contains the text to show in the block, the color of the block and the way it interacts with other blocks (a tool to create this definition using Blockly blocks to avoid syntactic mistakes is presented in Section 4.3). The translation is done in JavaScript for the interpreter and in Python to have a separate code (that will not contain any animation logic) for the tests in the backend of INGINious.

```

1 Blockly.Blocks['maze_moveForward'] = {
2   init: function() {
3     this.jsonInit({
4       "message0": "move forward",
5       "previousStatement": null,
6       "nextStatement": null,
7       "colour": Maze.Blocks.MOVEMENT_HUE,
8       "tooltip": "Move the pegman to the next square"
9     });
10  }
11 };
12
13 Blockly.JavaScript['maze_moveForward'] = function(block) {
14   return 'moveForward(\'block_id_\' + block.id + '\');\n';
15 };
16
17 Blockly.Python['maze_moveForward'] = function(block) {
18   return 'moveForward()\n';
19 };

```

Listing 4.1: Definition of the block used to move forward in the maze

Next, we can fill the zone for graphical content (created when the option “Visual” explained in Section 4.1.3 is enabled) with the skin of the maze and the animations have to be programmed. The code defining all the visual animations is too big to be shown here (1500 lines in its original version and still 800 in mine), but it is available at https://github.com/fthuin/INGInious_blockly_tasks. To go on with our example,

here are the specifications of a function:

```

1 /**
2  * Attempt to move pegman forward or backward.
3  * @param {number} direction Direction to move (0 = forward, 2 = backward).
4  * @param {string} id ID of block that triggered this action.
5  */
6 Maze.move = function(direction, id) {
7     ...
8 };

```

Listing 4.2: A function to move the character forward or backward

Afterwards, we have to tell the interpreter how to run the code generated by blocks. In the block definition of `maze_moveForward`, you can see that the generated JavaScript code will be of type `moveForward(<block_id>)`. In the interpreter, we create a new native function called `moveForward` and its code will be a call to `Maze.move` as shown in Listing 4.3.

```

1 var initInterpreterApi = function(interpreter, scope) {
2     var wrapper;
3     wrapper = function(id) {
4         Maze.move(0, id.toString());
5     };
6     interpreter.setProperty(scope, 'moveForward', interpreter.
7         createNativeFunction(wrapper));
8     ...
9 };

```

Listing 4.3: Example of native function added to the interpreter

So here are all the parts needed when you create a new blocks that have to be interpreted to be run locally. First, the block definition, then a function that should be called to animate a move and finally the link between the code generated by the block and the code to display the animation. The final result of the integration of the maze in INGINious is shown on Figure 4.7.

The only thing left is the testing of the code by the server, which can be done as for any other tasks with the information stored in the page as explained in Section 4.1.2.

4.3 BLOCKLY TASK CREATION TOOLS

INGInious tasks definition are stored in file called `task.yaml`. Those files are easy to be programmatically read and write but are not very human-friendly. To ease the creation and modification of tasks, the INGINious web application allows to modify that

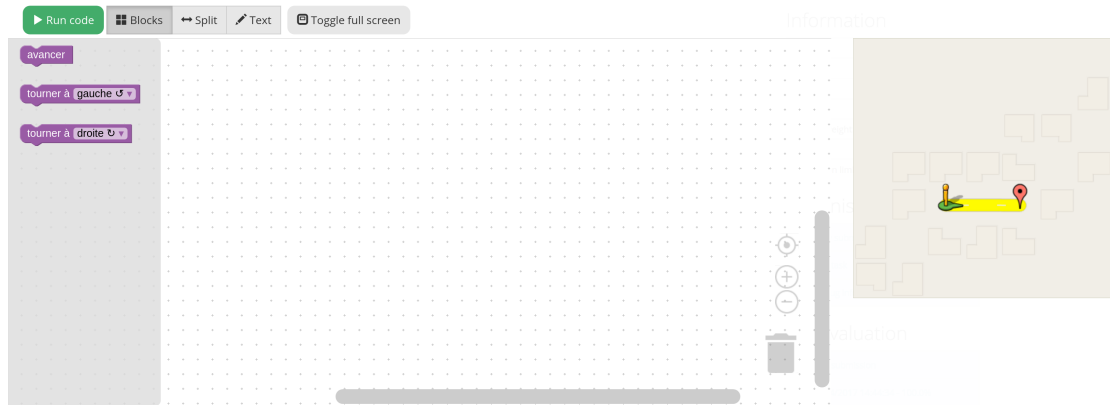


Figure 4.7: The Maze from Blockly Games integrated to INGINious

file from a [GUI](#). Basically, you can modify the text displayed for the problem as for other problems, but the customization the workspace and task through all the options described in [Section 4.1.3](#) is also implemented.

I’ve already cited the Blockly Developer Tools that allows to create new blocks using a Blockly [\[17\]](#). This is a very nice resource to avoid reading all the documentation or making syntax errors (it works like writing a program with Blockly). Blocks created in the Block Factory can be exported through the block exporter and the files created can be put into the related task directory. But the last feature of those Developer Tools, called “Workspace Factory” allows the user to choose the blocks displayed for the problem (toolbox and workspace) by using only Blockly workspaces. This feature is really interesting because creating the toolbox or blocks displayed at the loading of the task has to be done in XML and it is not very practical (e.g. the standard toolbox presented on the official website demo is defined by more than 300 lines of XML). Furthermore, the XML has to be written inside the `task.yaml` file, which is exactly what we want to avoid using the GUI. So something very similar to the Workspace Factory is integrated inside the task editor of INGINious for Blockly problem.

I could not reuse the code of the Workspace Factory, because there were some assumptions on the HTML framework used (which was not Bootstrap) but also because this demo is not up-to-date (some new blocks were missing). The Workspace Factory has a dependency to the Closure Library (which is composed of more than 2700 files) only because of its color picker. When you work with it, the data are shared inside files that you upload and download, but for the GUI of INGINious, data had to be updated directly in HTML elements. With all those differences, I preferred to rewrite it completely from scratch but providing the same layout and functionalities, as shown in [Figure 4.8](#).

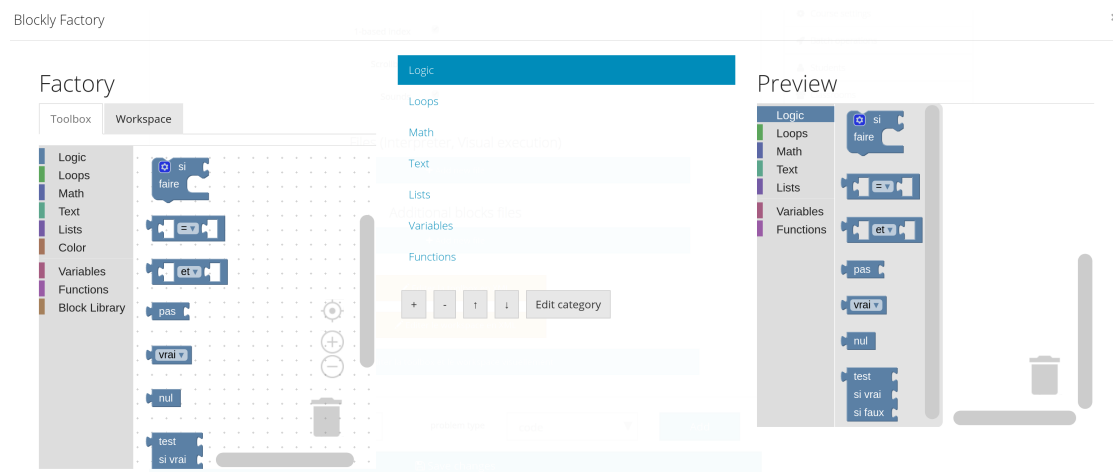


Figure 4.8: The Blockly Factory for an INGINious problem

This chapter describes the organization and material of activities where the implementation of Blockly in INGIInious was used to teach basic programming concepts to more than 150 kids (Section 5.1). Results of those activities are presented and discussed in Section 5.1.3. From those experiences, some advices are given to use the platform in a way that maximizes the learning outcomes of kids (Section 5.4–5.6).

5.1 PRINTEMPS DES SCIENCES

5.1.1 *Description*

During the week from March 20 to 26, UCL and Scienceinfuse organized an event called “Printemps des sciences” (Spring of sciences) to promote biology, chemistry, geography, geology, computer science, mathematics, physics and statistics for primary and secondary students.

One of the computer science activity proposed to teachers was made for 10 to 15 years old kids to discover programming. During the week, teachers come with their classes and during the weekend, any child can come (with or without their parents). Kids were encouraged to work by pairs on the same computer to help each other. The complete schedule of activities is shown in Table 5.1.

5.1.2 *Material*

I ran an instance of INGIInious on a server given by the INGI departement with the specifications shown in Table 5.2.

Two tasks were proposed to the kids: “Plant VS Zombie” and “Bee” (in this order, to provide an increasing difficulty). For the visual part, pictures were reused from tasks on Code.org [22]. For the code, the code of the Maze used to test the integration of Blockly in INGIInious (explained in Section 4.2) was reused with little modifications. Those modifications will be explained together with the tasks explanation.

The tasks are made to teach sequential execution, loops, conditions with Plant VS Zombie. For kids that are faster, tasks with function calls and functions writing were created in the Bee course but we don’t expect all kids to reach them because activities

Day	Hour	Class
Monday 2016-03-20	11.00-12.30	5th year of primary school
Tuesday 2016-03-21	11.00-12.30	5th year of primary school
Thursday 2016-03-23	09.00-10.30	6th year of primary school
Thursday 2016-03-23	11.00-12.30	1st and 2nd year of secondary school
Friday 2016-03-24	09.00-10.30	1st year of secondary school and newcomers program
Friday 2016-03-24	11.00-12.30	5th year of primary school
Sunday 2016-03-26	13.30-14.30	Kids from 10 to 15 years
Sunday 2016-03-26	14.45-15.45	Kids from 10 to 15 years

Table 5.1: Schedule of the activities

Component	Description
Processor	Intel(R) Core (TM) 2 Quad CPU Q6600 2.40GHz
RAM	4 × 2GB DDR2 (667 MHz)
HDD	250GB Seagate ST32500N
Network card	Intel 82566DM-2 Gigabit Network Connection
Linux kernel	3.10.0-514.10.2.el7.x86_64
Operating system	CentOS Linux release 7.3.1611
Graphic card	NVIDIA G86 (Quadro NVS 290)

Table 5.2: Workstation specifications (Sun Ultra 24)

are limited in time. Some of the next figures contains a mix of French and English but the buttons directly to Blockly were in French during the activities.

5.1.2.1 *Plant VS Zombie*

Plant VS Zombie is a game that has a lot of success (more than 1 million downloads for its paid version [23]) where players have to use plants to prevent an army of zombies from entering houses and eating their brains [24].

For the task, the theme of the game is reused, but the gameplay is different. Kids are playing as a Zombie that has to find a path to reach and eat a sunflower, avoiding to walk on carnivor plants. To reach that goal, they have to program the movements of the zombie with Blockly blocks. The thirteenth task of the Plant VS Zombie course is shown in Figure 5.1 and a complete list of concepts addressed in each task is shown in Table 5.3.

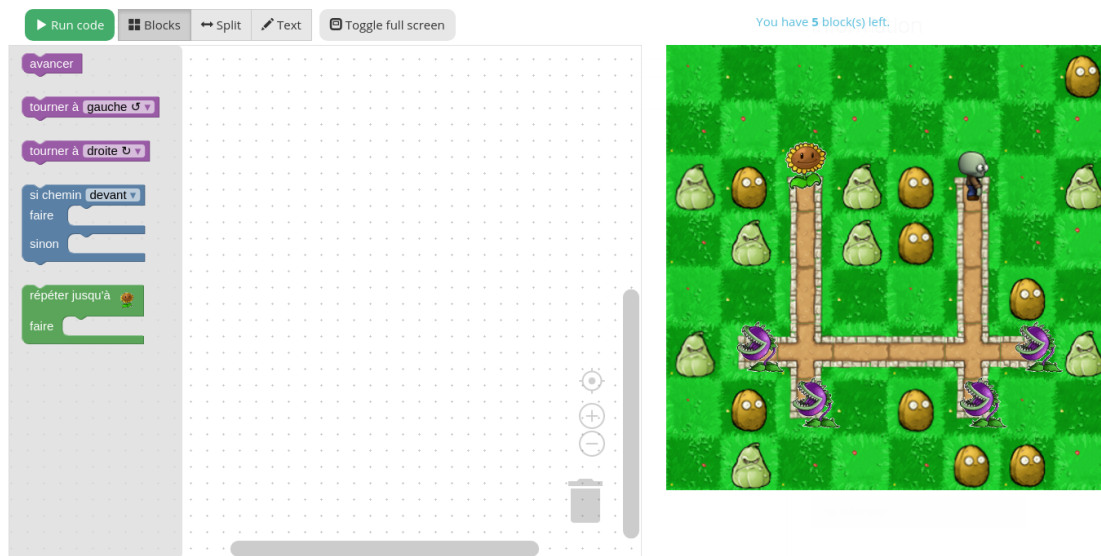


Figure 5.1: Example of Plant VS Zombie problem

Plant VS Zombie tasks are different from my adaptation of the Blockly Games maze (Section 4.2) by only one aspect (excluding the visual) that is the existence of obstacles. So the logic related to the obstacles were added to animate the failure during client-side testing (if the zombie walks on a plant, the plant eats the zombie and no other actions can be done) and in the backend testing (by adding a condition of success that is avoiding walking on a plant).

5.1.2.2 Bee

Bee is a game where a bee has to collect nectar from flowers and can make honey from it. The bee can collect nectar a limited number of times on each flower and make honey a limited number of times in each hive. Kids have to code the movements and actions of the bee to collect the maximum number of nectar and to produce the maximum of honey. An example of the last problem is shown in Figure 5.2, a complete list of concepts addressed in each task is in Table 5.4.

Task number	Concepts
Task 1	Sequential execution
Task 2	Sequential execution
Task 3	Sequential execution
Task 4	Sequential execution, for loops
Task 5	Sequential execution, for loops
Task 6	Sequential execution, for loops
Task 7	Sequential execution, for loops
Task 8	Sequential execution, for loops
Task 9	Sequential execution, for loops
Task 10	Sequential execution, while loops
Task 11	Sequential execution, while loops
Task 12	Sequential execution, while loops
Task 13	Sequential execution, while loops, if-else conditions
Task 14	Sequential execution, while loops, if conditions
Task 15	Sequential execution, while loops, if conditions

Table 5.3: Concepts addressed in each task of the Plant VS Zombie activity

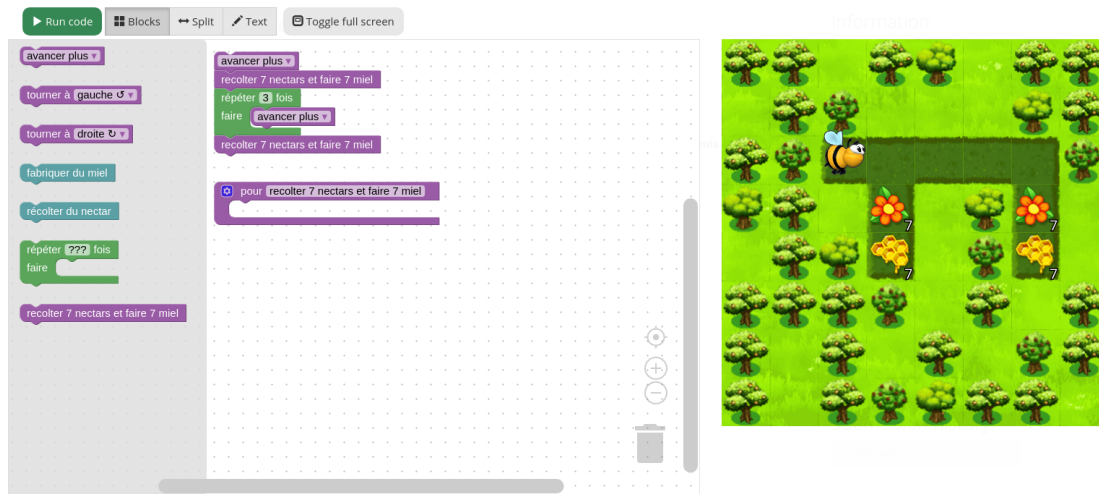
Here, modifications of the code from my implementation of the Blockly Games maze were made (in the client-side and the backend tests) to change the success verification of the termination. Previously, it was a position to reach but here, the problem is resolved if the number of resources left is equal to 0.

5.1.3 *Analytics*

This section will describe the flow of students resolution during each activity and their feelings about the activity at the end of it.

5.1.3.1 *First activity*

During the first activity, there were 21 students who formed 11 pairs. As you can see on Figure 5.3, all students reached the last task of the Plant VS Zombie course although

Figure 5.2: A problem from the *Bee* course

Task number	Concepts
Task 1	Sequential execution, for loops
Task 2	Sequential execution, for loops
Task 3	Sequential execution, for loops, function calls
Task 4	Sequential execution, for loops, function calls
Task 5	Sequential execution, for loops, function writing
Task 6	Sequential execution, for loops, function calls
Task 7	Sequential execution, for loops, function writing
Task 8	Sequential execution, for loops, function writing

Table 5.4: Concepts addressed in each task of the *Bee* activity

one student forgot to submit the 9th task resolution. Figure B.1–B.9 show the answers to the survey. Almost all students would like to do that kind of activity at school or at home again. The worst results were for the principle of conditions, the time allowed for the activity and the interface. For the conditions, it can be explained by the lack of activities containing conditions and that some activities could be solved without using them. For the time allowed, it is normal because nobody reached the end of all tasks. For the interface, there were several problems:

- There wasn't any animation when they succeeded (so some of them kept searching for a solution). In the next days, zombie will eat the sunflower and laugh when they walk on it;
- A button was missing to reinitialize the task. Concretely, there were only two buttons for the test: start and stop. So if you stopped the execution, the character stayed on its place and students cannot reset it to the original position (unless they push start again, but it reexecuted the code immediatly). For the next days, a button to restart at initial position after stopping was added.
- INGINious interface is in English and I forgot to translate some buttons so it was hard to explain them where to click;
- The task loading time was very long (because the task loads mp3 files, pictures and javascript files from the server that has an old configuration) and I did not load the first task before they arrived so when they all accessed the task together it took a lot of time to display all resources. By default with INGINious, There are no cache options for file retrieved in the public directory of a task, so at each task every resources had to be reloaded.

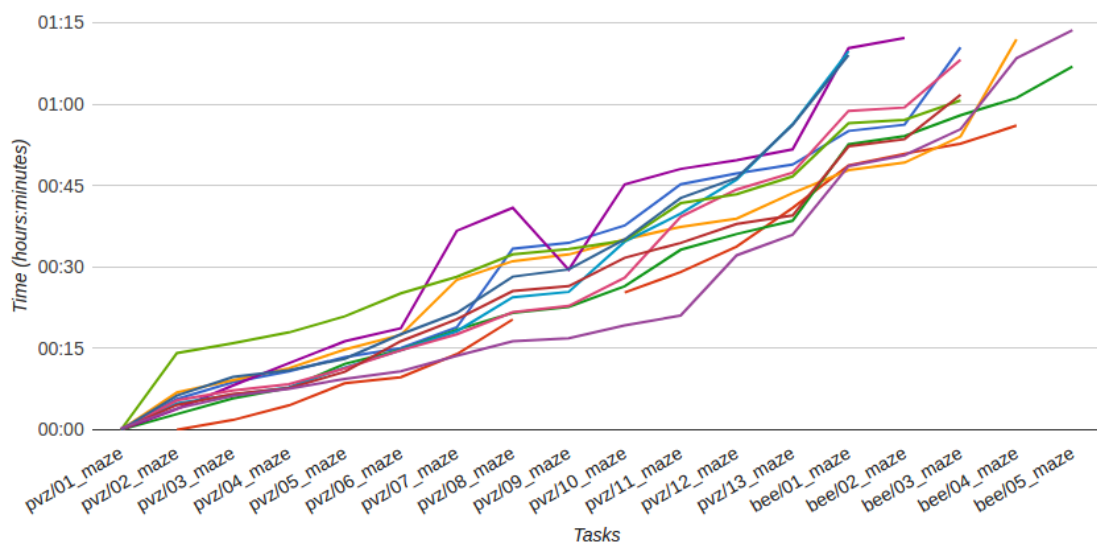


Figure 5.3: Tasks completed over time by each pair of students (March 20, 2017)

5.1.3.2 Second activity

For the second activity, there were 24 students. Students used 14 computers because some of them preferred to be alone and one of the teacher also tried the first tasks and

stopped at the 9th. As you can see on Figure 5.4, only 2 out of the 14 groups did not reach the last task of the Plant VS Zombie course, which means they didn't tried a task with conditions.

As you can see on Figure B.10–B.18, everyone understood the principle of sequential execution, loops and functions and receive enough help from us to understand the tasks. Conditions seems harder to understand for kids. A large majority of them would like to do it again at school or at home.

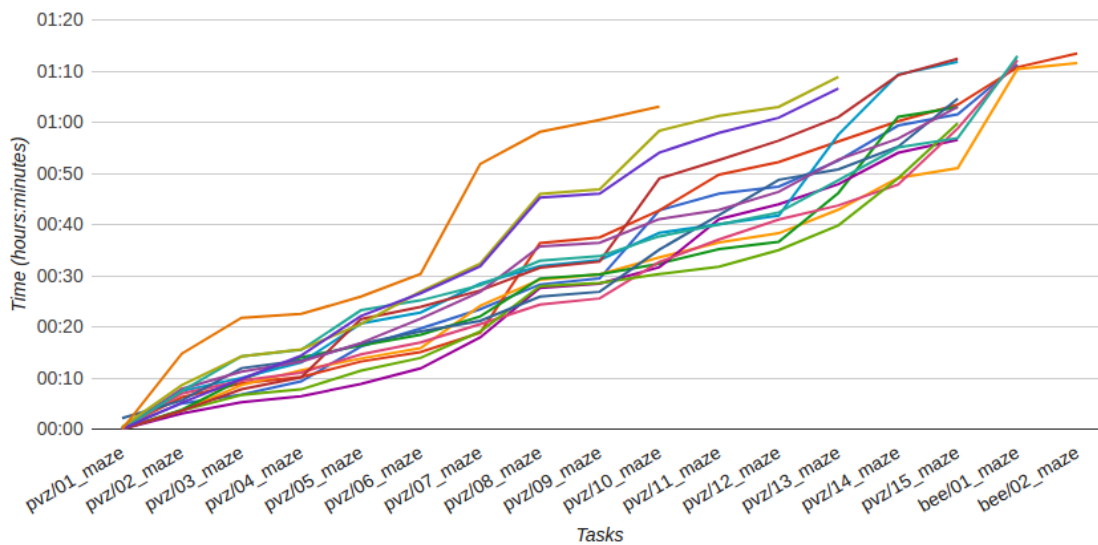


Figure 5.4: Tasks completed over time by each pair of students (March 21, 2017)

5.1.3.3 Third activity

For this activity there were 24 students and who formed 13 groups. Unfortunately, three groups did not reach the end of all tasks and one of them did not reach the concept of condition, as shown in Figure 5.5. This time, kids liked less the interface and they less want to solve problems using Blockly at home or at school even if there is still a majority of them who would like to.

5.1.3.4 Fourth activity

During this activity, 19 students of first and second year of secondary and their 3 teachers formed 11 groups. As you can see on Figure 5.6, the groups had really different progression. The one who started latest is a group of two teachers. The third teacher is a teacher of electronics who already taught his second year of secondary students C++ which explains why some groups nearly reached the end of the second course. The

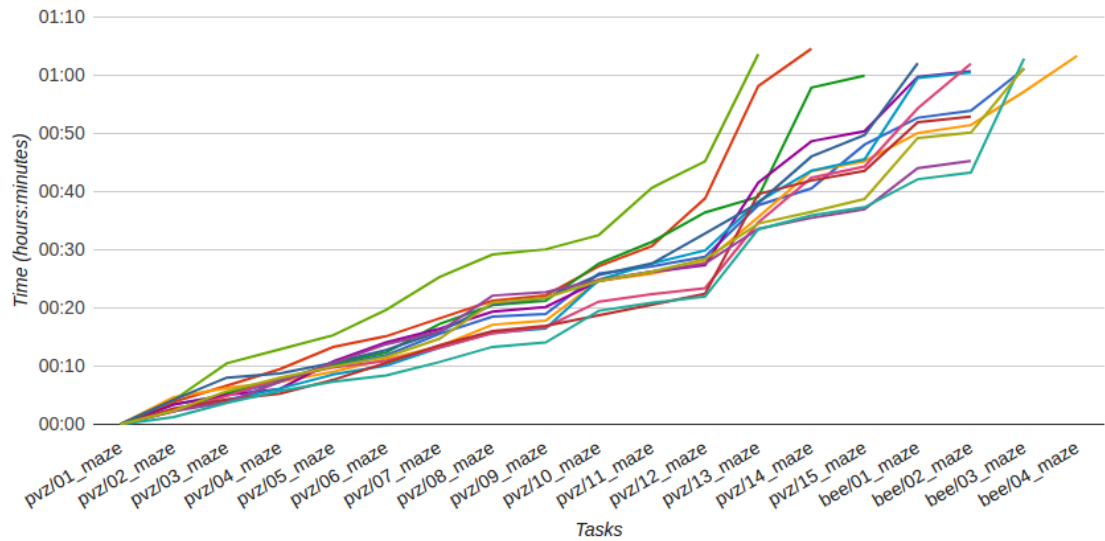


Figure 5.5: Tasks completed over time by each pair of students (part 1 March 23, 2017)

students of first year did not had any course yet so their progression is alike the one of other days. As you can see on Figure B.28–B.36, they all understood the principles of sequential execution, conditions and loops. A large majority of them understood the principle of functions and would like to do it again at school or at home.

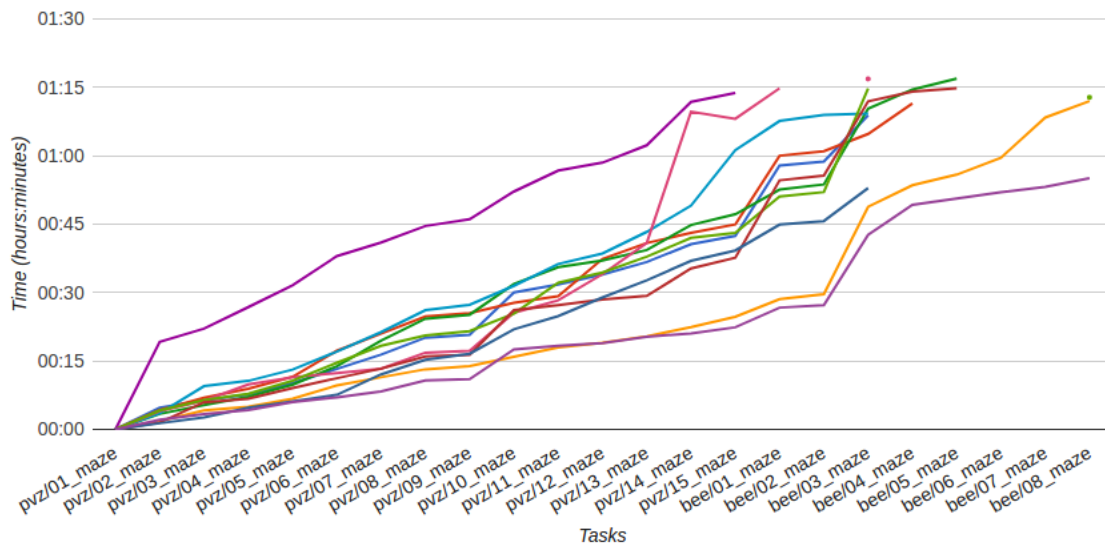


Figure 5.6: Tasks completed over time by each pair of students (part 2 March 23, 2017)

5.1.3.5 Fifth activity

The public of the fifth activity was different, there was a class of first year of a secondary school (14 students) and 5 newcomers who formed 10 groups. As the newcomers don't speak very well French, it was hard for them to understand the explanations of the concept or the text on the blocks or the interface. As shown on Figure 5.7, only 60% of the groups reached the end of the first course. This is worst result for the 1h30 activities, and as shown on Figure B.37–B.45 half of them would not like to do it again at home or at school. Despite those results, there is still 80% of the participants saying they understood the basic principles of programming, but it means that the exercises and my explanations didn't manage to give them the motivation to do the exercises nor made them really interested by programming.

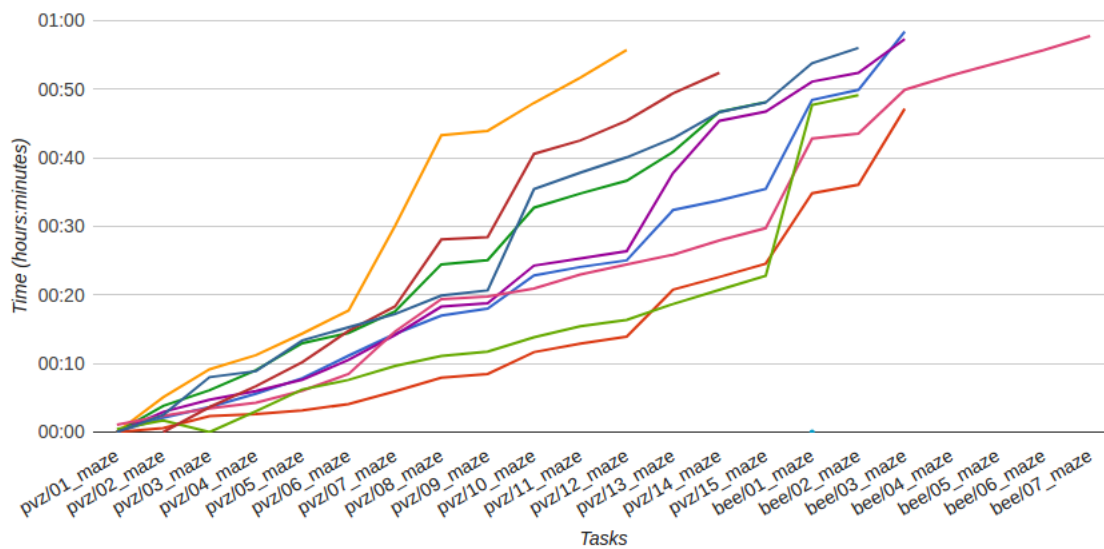


Figure 5.7: Tasks completed over time by each pair of students (part 1 March 24, 2017)

5.1.3.6 Sixth activity

23 students participated to the sixth activity in 12 groups. Every group reached the concept of conditions and only three did not reach the end of the first course, as shown in Figure 5.8. More than 80% of the kids understood the principle of sequential execution, loops and functions, they liked the interface and would like to do this kind of exercises again at home or at school, as shown in Figure B.46–B.54

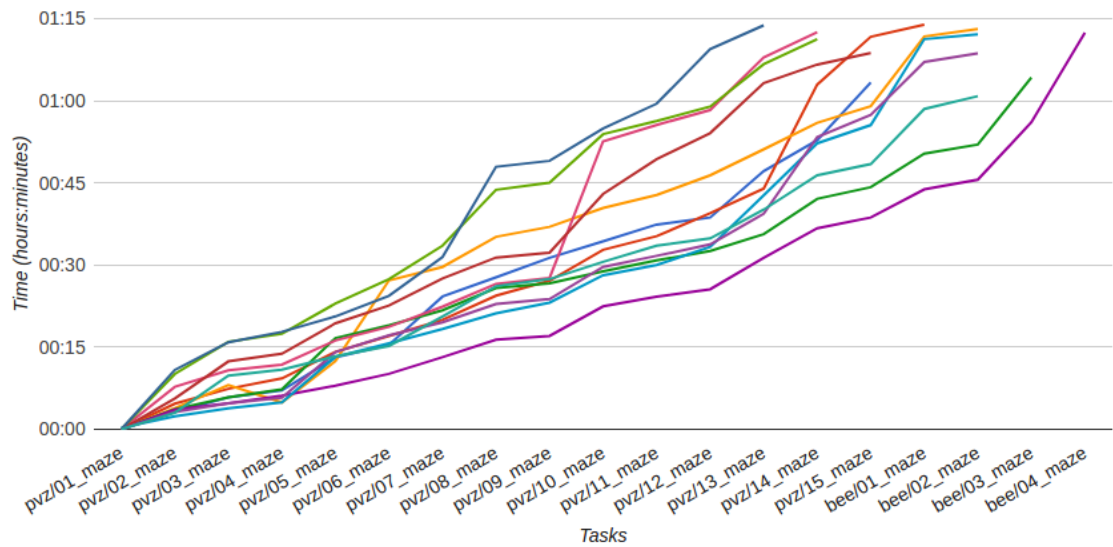


Figure 5.8: Tasks completed over time by each pair of students (part 2 March 20, 2017)

5.1.3.7 *Seventh activity*

For this activity, there were 11 groups of students. Kids came late so as there was only one hour of activity, I preferred to take more time to solve the problems and I didn't ask them to complete the survey. As you can see on Figure 5.9, only two groups reached the end of the first course. 40 minutes is a bit short to help them learn at least the principle of sequential execution, loops and conditions. I think that at least one hour of time is necessary to ensure that all motivated students can try once to solve a problem containing each concept.

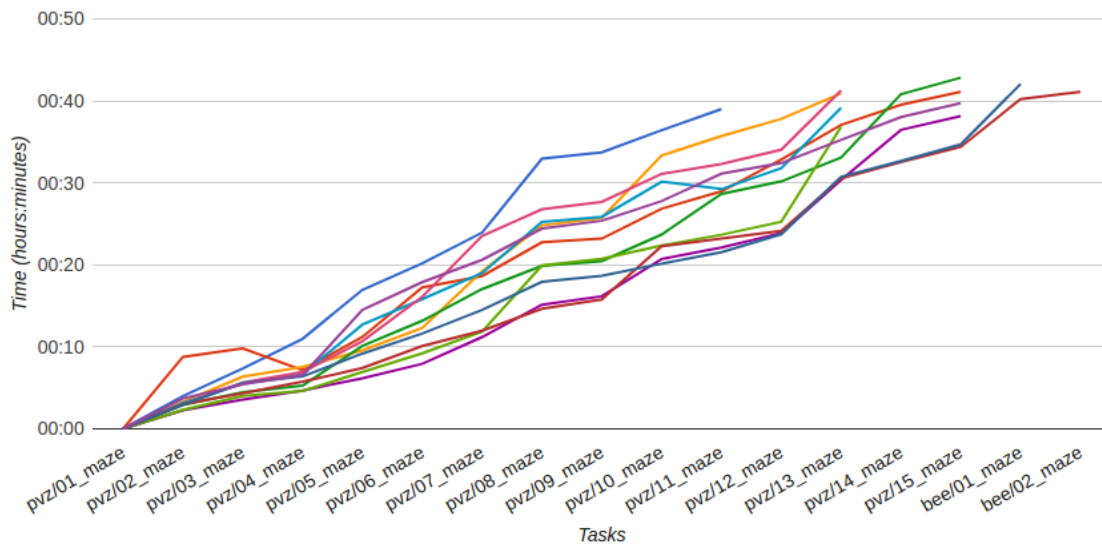


Figure 5.9: Tasks completed over time by each pair of students (part 1 March 26, 2017)

5.1.3.8 Eighth activity

During this activity, there were 10 groups of kids, some accompanied by their parents. As you can see on Figure 5.10, the tasks completed over time looks weird for two of them. The purple one that started directly with the tasks of the Bee course because she came with her school during the week and wanted to try to reach the end (so she was alone), the light blue one was her little sister (7 years old) that also tried the exercises alone. This was really nice to see that some kids really liked the activities enough to come back and try it again, it shows that this work might really bring interest for programming to young people.

All kids reached at least the first task that contains condition. As you can see on Figure B.55–B.63, if we exclude the girl that was too young to participate to this activity, every kid received enough help, would like to do it at school, find the interface easy to use and understood the principle of loops and sequential execution. Only one group says they didn't understand the principle of conditions and functions and didn't want to do it at home.

5.1.3.9 Summary

The results of the activities are pretty nice: they show that a single person can manage alone a class of kids (the two first activities were supervised by Olivier Goletti, but after it, I felt comfortable enough to do it alone and the results are still good) to teach programming. Even if we let kids learn by themselves at their own rhythm, the class

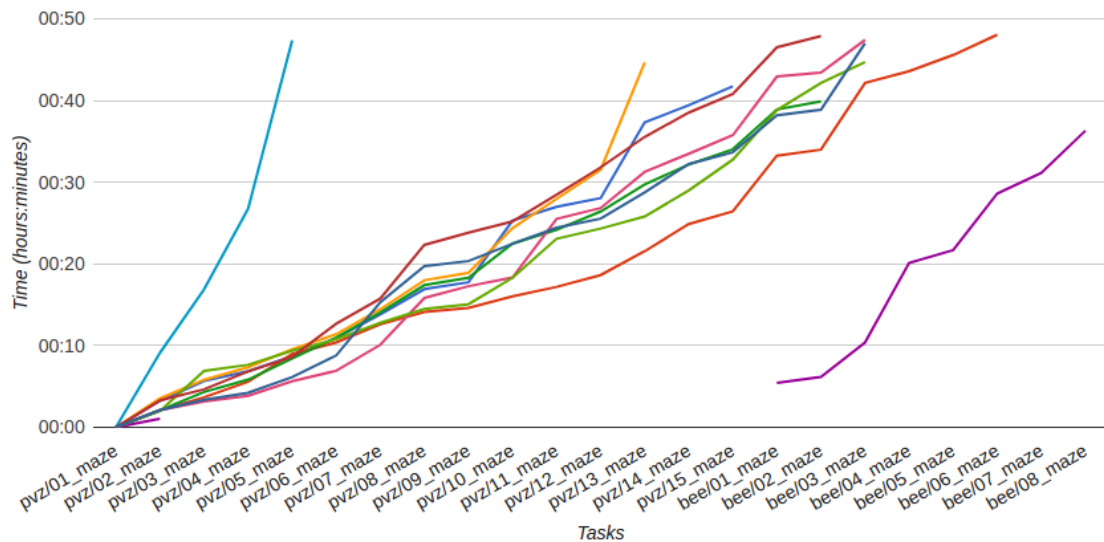


Figure 5.10: Tasks completed over time by each pair of students (part 2 March 20, 2017)

is usually quite homogeneous in its progression. Most kids liked that introduction to programming enough to have the desire to continue it at home or at school.

There are still improvements that are needed on the interface to be more friendly to kids, but as the results show, as the week progresses the kids seem to like it more. This can be explained by the fact that I took more time to explain how it worked after seeing the main difficulties of kids. The main difficulty encountered was to know how to switch between tasks and how to switch between courses (they didn't look used to navigate into a website).

5.2 HOW TO TAKE ADVANTAGE OF BLOCKLY ON INGINIOUS

On Code.org and other sites of the same kind, the tests made to ensure that a task was passed are all in the scripts given to the user. First, this technique will give all the tests to the user (this is not a major drawback because the user is not supposed to understand test code) but it will also run all the tests on the client side. If you do a very big exercises which needs a lot of tests (for example if you use randomness somewhere in the problem) and the client machine has limited resources (CPU, RAM, battery, ...), doing all the tests on a server is far more convenient.

The other advantage is that you can have a feedback (as shown in Section 5.1.3) that can help to improve the course, to know the progression of each student and their comprehension of the concepts. This is interesting to improve a task for differ-

ent classes, but we can imagine that if a teacher starts to use INGINious with his class, he can use it to go back to the principles that were hard to understand for his students.

With Blockly on INGINious, you can provide two different types of feedback: one generated from tests run in the client-side and the other from tests run in the backend. This is powerful because if you imagine that students have to be graded from an exercise of Blockly, you can give them a visual representation of the kind of problem you want them to solve (to allow to test their code with a sample) but in the backend you can run tests on different situation and give them a feedback (for example a picture) of a situation where their code does not provide a working solution. This is interesting if you want them to start thinking in a more abstract way (in which they have to think by themselves about all possible situations).

5.3 WHEN TO AVOID BLOCKLY ON INGINIOUS?

Basically, Blockly on INGINious offers the same as any other educative platform if you don't want to see the productions of the students, if you don't need the classrooms, if you know that the client machine is powerful enough to run any kind of tests, if you don't want to bring any abstraction to kids or if you don't want them to access their older submissions.

For the moment, as this work is the first one for Blockly on INGINious and I was alone to produce the content, other platforms have a wider range of content and a community making them more interesting. INGINious does not yet support localization to work with different languages (the platform is only available in English for the moment, although I translated it for the duration of the activities). The support for additional languages is under development [25] and the second edition of the Open Week event is organized in July 2017 and proposes workshops about Blockly [26], so more content might be created and a community of developers for Blockly on INGINious might start existing in the near future.

5.4 BEST PRACTICES

This section discusses the best practices that have been used and should be used. Tasks will be introduced, then their content will be shown and afterwards there will be a discussion on the feedback of the test.

5.4.1 *Problem presentation*

Each problem should have short explanations (most kids will probably not read it unless you force them to, but none will if it is too long). There should not be too many elements in the interface and they must be grouped by their actions. Each button action must be clearly explained to all students at the moment they need to use it.

The best way to start the activity is to first log in yourself all students in the platform, to put the browser in full screen mode (to avoid them to close it or to click on buttons outside the platform) and to load the page of the first problem before they arrive. This will allow you to start the activity by presenting them the activity and by asking them how to solve the first problem. Once a solution has been found, you show them the toolbox to explain the blocks and you ask them to drag-and-drop blocks into the workspace to build their solution, you need to be clear with the fact that blocks should be connected to each other from top to bottom (some kids tried to put them side-by-side). Once it is done, you can ask them to press the button “Test” to show that the code can be executed to see the result on the visual part. Once all students have successfully done it, they can click on the submit button, wait for the green alert that says the success. Tell them that if the alert is not green in this task or the following ones, it is because there is a problem and they can ask for help.

Make them click on the course name on the top navigation bar to see all tasks and tell them that they can click one the next one and do it at their rhythm. Repeat to them that they need to use the “submit” button at the end of each task and to have the green alert to switch to the next task. For the students that are faster, introduce the concepts to them and once you have a certain number of students asking simultaneously for explanation of the concept, ask for silence and explain it to the whole class. You need to stop everyone (even those who already know it) to introduce the usage of the block but above all the underlying concept.

If by any chance, you have the chance to work with touch screen computers (as it was the case for some of the activities), do not hesitate to tell the kids that one kid of each group should work with the mouse and the other should help with the touch screen. Kids that work in group will thus be able to interact with the interface at the same time to help each other.

5.4.2 *Problem content*

The problem content should introduce at most one concept. The idea is to introduce a concept, then repeat this concept in different problems, then introduce a new concept and use it with and without the previous concept (if possible) and so on. Try to intro-

duce a gradually increasing difficulty, do not hesitate to make a very easy exercise first to make the kids confident and to end with exercises that requires a lot of thinking to be solved.

The content should use words that any kid understand, but it should also use concept that are practical. For example, if you are not doing it for mathematical reason, don't use "turn to 90, 180, 270 degrees" but use 3 blocks, turn right, turn left and turn around or a block *turn* with those parameters. For blocks, use preferably only infinitive verbs that have a single meaning.



Figure 5.11: Plant VS Zombie task 6

Do not hesitate to create problem that can be solved with different solutions (one that is simple and another that requires more thinking), this will allow you to challenge the students that are faster. For example, the sixth problem of Plant VS Zombie shown on Figure 5.11 can be solved with 8 blocks using simple loops (shown in Figure 5.12), but 4 blocks with the concept of imbricated loops (shown in Figure 5.13).

5.4.3 Test feedback

When you write feedback, you can do it whether during the local tests, or after the backend tests or both. Those two feedbacks do not have the same content, during a local test, the best is to display a very short sentence (because kids will have to read it at the same time as they look at the visual animation) to say the rule that was not respected.



Figure 5.12: Solution to task 6 of Plant VS Zombie using simple loops



Figure 5.13: Solution to the task 6 of Plant VS Zombie using imbricated loops

In the backend feedback, you need to contextualize the situation in which a problem occurs. Do not hesitate to be verbose by showing a predefined example of such errors or by writing multiple sentences to explain the problem. Kids will pay attention to it because you (should have) told them that they cannot go to the next problem if they don't have a green alert that say they have succeeded.

5.5 MISTAKES TO AVOID

During the first activity, there was no limitation in the number of blocks that could be dragged in the workspace, leading to students passing tasks without using the concept newly introduced. As the tasks gradually become more difficult to solve, they get stuck because they didn't use the concept on a simple problem. All problems have to be limited in number of blocks to ensure that students will be forced to use the concept at least once.

Never try to explain something to a kid that has a mouse in his hands. He will not be concentrated for your explanation or at least not until the end. Do not hesitate to put yourself as if you were the character they are trying to code to keep his attention out of the screen. Usually, when a kid is stuck on a problem, it is because he does not visualize the effect of each of the blocks and if the execution on the screen failed to help, it is preferable to explain it from a completely different point of view.

5.6 SUMMARY

Blockly in INGINious allows one person to teach programming concepts to a full class of kids, making a vast majority of them understanding and succeeding tasks that requires the concepts of sequential execution, loops and conditions in activities of approximately one hour. Kids participating to activities with Blockly on INGINious showed an interest in doing it again at home or at school.

CONCLUSION

6.1 RELATED WORK

6.1.1 *BlockPy*

One of the most interesting project I discovered during my researches was BlockPy [27], it is a Python environment designed for Data Science that lets you work with text, blocks or **both**. And by *both*, it really means both: you can use blocks and it will be translated to Python (as it is done here on INGINious), but if you write Python without syntax errors, it will parse it to get an abstract syntax tree and transform it into blocks. It is developed at Virginia Tech and it has the main developer of Blockly as contributor. It also allows to see step-by-step execution with values of variable at each step. Those features are really interesting if we want Blockly on INGINious to serve as a springboard for kids to learn to code with a real-world language.

6.1.2 *Code.org*

The most advanced project to teach programming concepts to kids using Blockly is the platform Code.org, it does not share the same technologies as INGINious but the architecture of the library to create new problems is well structured and should be inspiring if we want to develop more content for Blockly on INGINious.

6.2 FURTHER WORK

During the summer holidays, the second edition of an event called the *Open Week* will be organized. This event is made to develop more content for INGINious and to make computer science students discover the world of open-source projects. This year, Blockly on INGINious is one of the topic proposed to the participants. It is an occasion to create a community of developers to provide more content on the platform and to improve the existing interface.

6.2.1 *Blockly for INGINious*

Most tasks created for Blockly share a common logic (visual animations, movements blocks, ...) thus it would be interesting to build a library with different abstract classes for common tasks to facilitate the creation of new tasks.

6.2.2 *INGInious*

INGInious is lacking some features, for example navigating to the previous or next tasks of the same course, hiding the panel that show information on a task page (to win some space on the screen). It was not made to use it for activity with supervision so there are no real-time information about the progression of students (this could help of the teacher to know the progression of its class without having to keep an eye on each screen). Support for multiple languages (at least English and French) is also very important to reach kids from Belgium. It would also be interesting to provide a way to connect to the existing platform (that is currently limited to UCL students) to any kid wanting to learn programming with Blockly on INGINious by themselves.

6.3 CONCLUSION

INGInious, an education platform that provides automatic code grading to assess programming assignments in a secure environment at Université catholique de Louvain was extended to provide content for kids. This content has been brought thanks to the usage of a visual programming language, Blockly. This thesis work regroups:

1. An overview of the existing visual programming languages aimed to kids;
2. A description of the tools used, namely INGINious and Blockly;
3. A high-level description of the implementation that allows to integrate Blockly problems inside INGINious;
4. A discussion of the results of activities organized with more than 150 kids and the best practices learned from this experience.

First, I investigated the different visual programming languages to choose the one that was the more convenient to integrate in INGINious. The choice went for Blockly because it is the only visual programming language available on the web that was made to be integrated inside another platform and that provides a library to communicate between its interface and the platform that contains it. It is also the one that was most able to take advantage of the automatic code grading feature provided by INGINious.

Once the choice of visual programming language was made, I integrated it to the existing implementation of INGINious, avoiding to modify any of its existing features. Simple content without visual representation of the problem was first added, followed by content where the solution can be executed in the web browser to display a visual representation of the problem and animate it, using an interpreter. The blocks can be translated to a textual programming language thanks to the Blockly library, and this translation was used to test the program with INGINious.

The implementation was tested and improved thanks to activities organized during the Printemps des Sciences with more than 150 kids. Two tasks were proposed to help kids learn core programming concepts: sequential execution, loops, conditions and functions. Two courses were created to introduce those concepts and the platform was proven to allow a single teacher to make a large majority of kids work by themselves to use and understand those concepts. The activities helped to draw the outlines of the best practices and the common mistakes faced when teaching programming to kids with a platform.

From a personal point of view, this work was rewarding because I found it nice to share my passion with kids. It was challenging because it is not an easy work to integrate a project inside another, and to make the result of the implementation accessible to kids. It was also challenging to teach programming to a class of kids because it is not something that we learn in our computer science courses and I had the chance to “learn it by doing”.

Part II

APPENDIX

APPLICATIONS BASED ON BLOCKLY

Blockly blocks work nicely for a wide range of domains. They are used in a lot of applications, including:

BLOCKLY GAMES: Little games to introduce the usage of Blockly to kids

APP INVENTOR: Creation of Android Application using drag-and-drop and blocks

CODE.ORG: Educative website using Blockly to make characters interact with their environments in a visual part of the screen, widely used for the *Hour of Code* event.

OZOBLOCKLY: Blockly is used to program the OzoBot, a robot with sensors and motors, design for 6+ years old kids.

MICRO:BIT: A collaboration between the UK, the BBC and Microsoft to offer a free pocket-sized codeable computer with many sensors to children in the UK and a website to code programs for it.

CODEBUG: A wearable device powered by a watch battery built after a Kickstarter project.

WONDER WORKSHOP: A project where two robots (Dash and Dot) can be used alone or to interact with each other (you can write code to control Dash from the buttons on Dot).

OPEN ROBERTA: A website to write programs using Blockly for multiple type of robots (NAO, micro:bit, Bot'n roll, Calliope, NXT, EV3) with a simulator for each of them.

MADE WITH CODE: A Google platform to help encourage the passion of science to teen girls.

ALL CAN CODE: A website made for kids to learn programming with cartoons characters that you help to do their adventures.

LEARN TO MOD: A website made for kids to create mods for Minecraft (29.99\$/year), developed by Multi-Dimensional

GAMEFROOT: A web-based application to program 2D games and share it (everything is stored in the cloud).

GAMEBLOX: A web application to make small 2D games developed by the MIT.

SCRIPTR: A web application for companies that want to build scripts for IoT, Social networks, . . . or to let their customers build them.

NOVA PBS: Games related to cyber security, new technologies and tech enterprises.

CODE SPELLS: A 3D game where you can create magic spells with code, developed by Multi-Dimensional.

THYMIO: A moving robot with multiple sensors and buttons that can be programmed.

RESULTS OF THE SURVEY

B.1 FIRST ACTIVITY

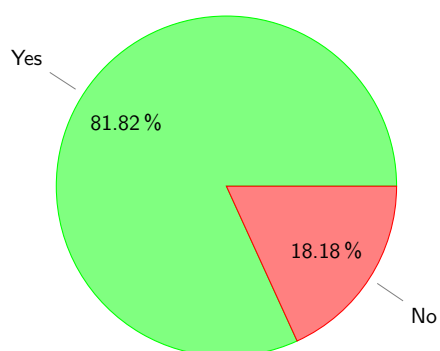


Figure B.1: "Would I have needed more time to do the exercises?"

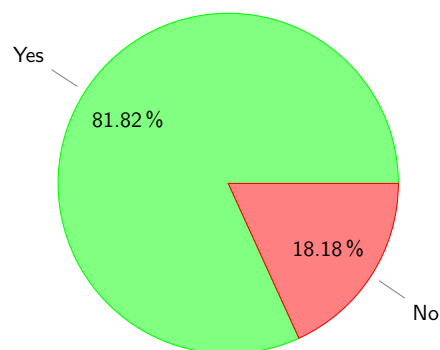


Figure B.2: "Did I understand the principle of sequential execution (execution of the blocks one after the other)?"

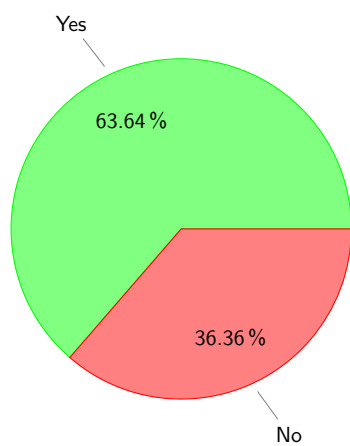


Figure B.3: "Did I understand the principle of conditions? (If ...do ...otherwise)"

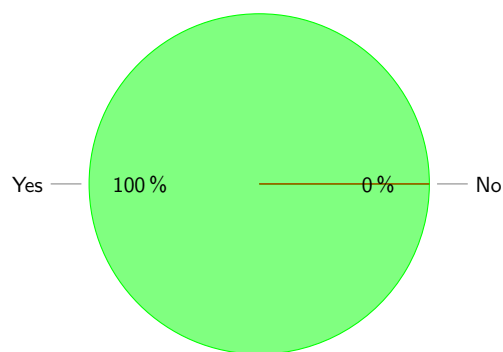


Figure B.4: "Did I understand the principle of loops? (repeat ...)"

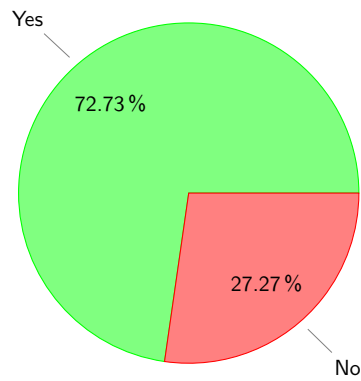


Figure B.5: "Did I understand the principle of functions? (Set of blocks with a name)"

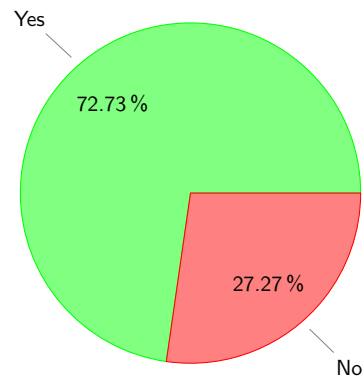


Figure B.6: "Was it easy to use the interface? (Move the blocks, test the code, ...)"

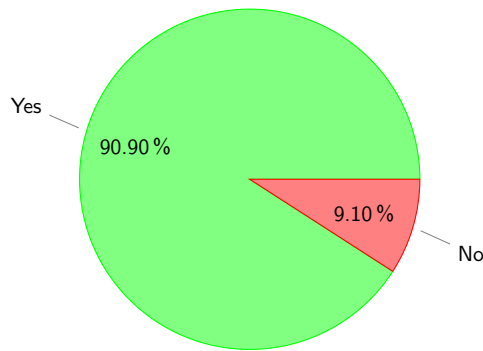


Figure B.7: "Do I want to do other exercises (games to solve with blocks) at home?"

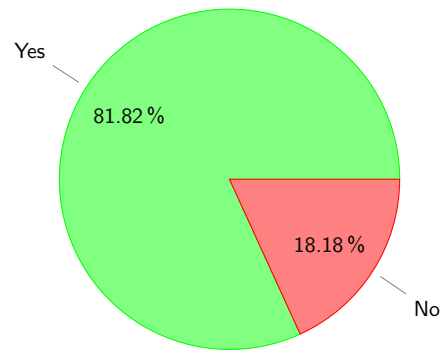


Figure B.8: "Do I receive enough help to be able to understand the tasks?"

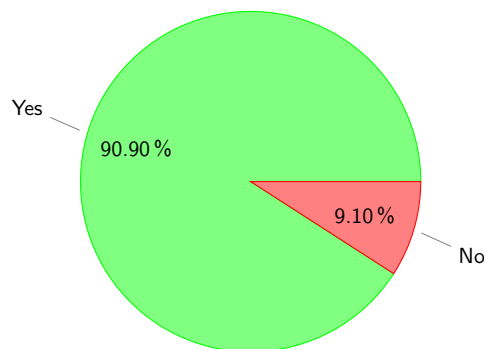


Figure B.9: "Would I like to do this at school?"

B.2 SECOND ACTIVITY

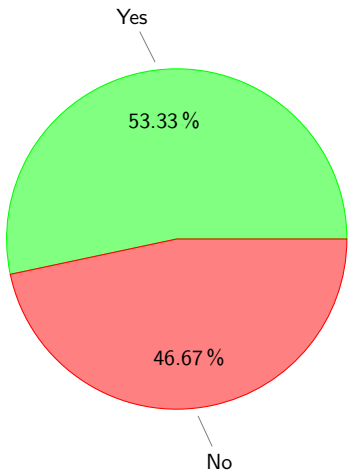


Figure B.10: “Would I have needed more time to do the exercises?”

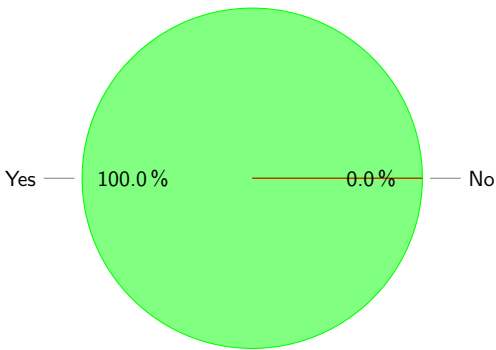


Figure B.11: “Did I understand the principle of sequential execution (execution of the blocks one after the other)?”

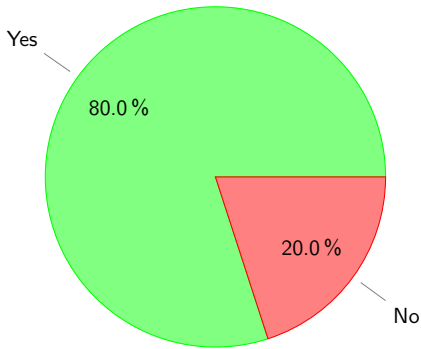


Figure B.12: “Did I understand the principle of conditions? (If ...do ...otherwise)”

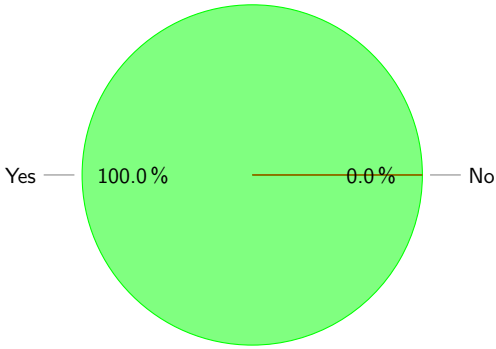


Figure B.13: “Did I understand the principle of loops? (repeat ...)”

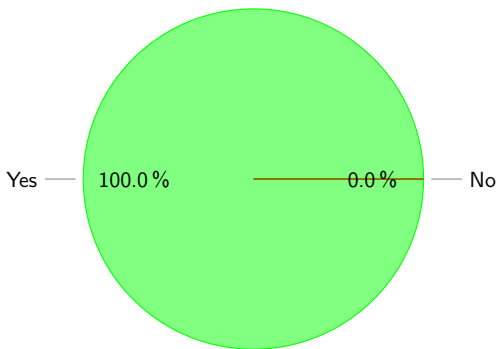


Figure B.14: “Did I understand the principle of functions? (Set of blocks with a name)”

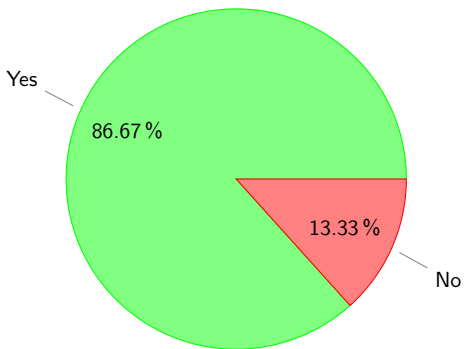


Figure B.15: “Was it easy to use the interface? (Move the blocks, test the code, ...)”

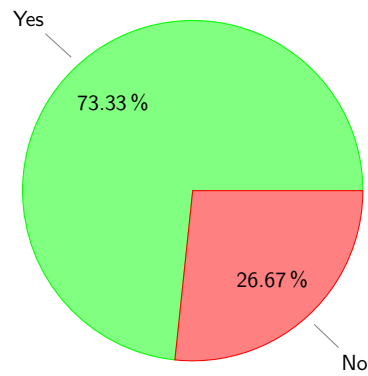


Figure B.16: "Do I want to do other exercises (games to solve with blocks) at home?"

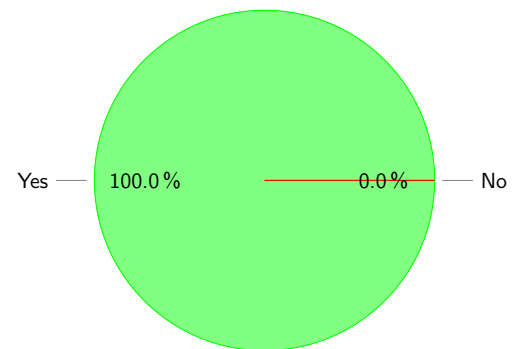


Figure B.17: "Do I receive enough help to be able to understand the tasks?"

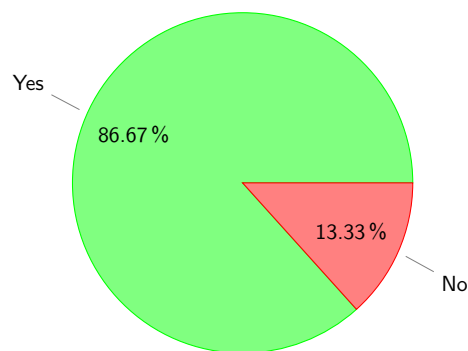


Figure B.18: "Would I like to do this at school?"

B.3 THIRD ACTIVITY

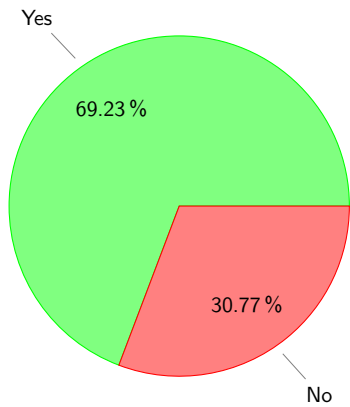


Figure B.19: “Would I have needed more time to do the exercises?”

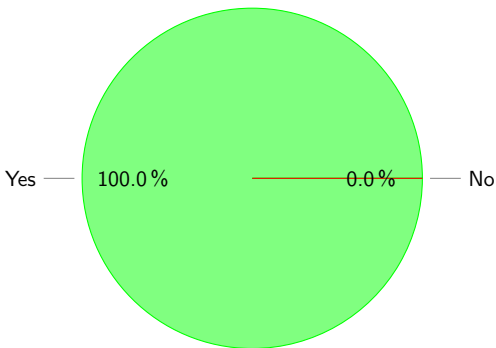


Figure B.20: “Did I understand the principle of sequential execution (execution of the blocks one after the other)?”

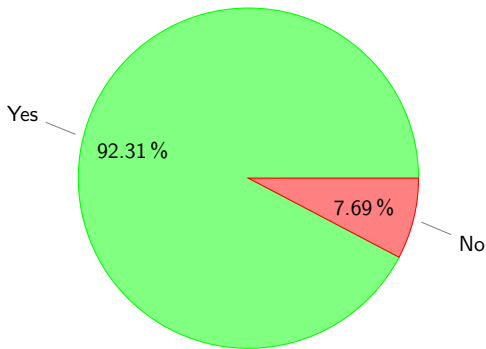


Figure B.21: “Did I understand the principle of conditions? (If ...do ...otherwise)”

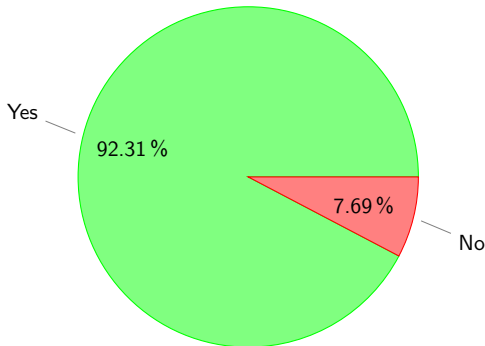


Figure B.22: “Did I understand the principle of loops? (repeat ...)”

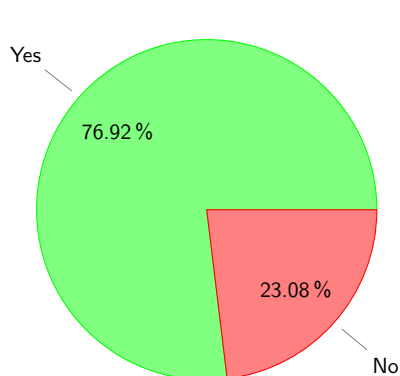


Figure B.23: "Did I understand the principle of functions? (Set of blocks with a name)"

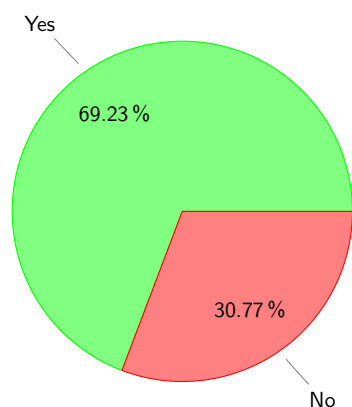


Figure B.24: "Was it easy to use the interface? (Move the blocks, test the code, ...)"

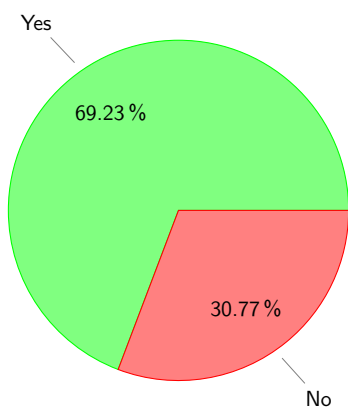


Figure B.25: "Do I want to do other exercises (games to solve with blocks) at home?"

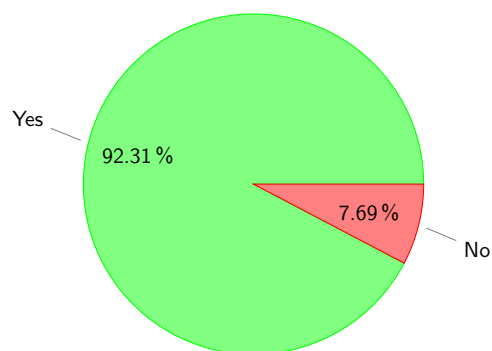


Figure B.26: "Do I receive enough help to be able to understand the tasks?"

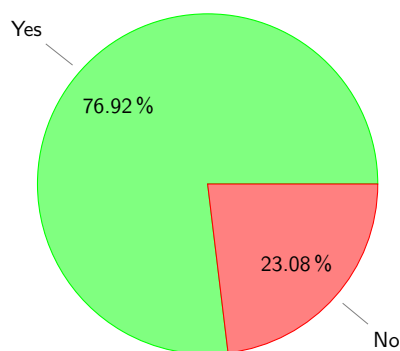


Figure B.27: "Would I like to do this at school?"

B.4 FOURTH ACTIVITY

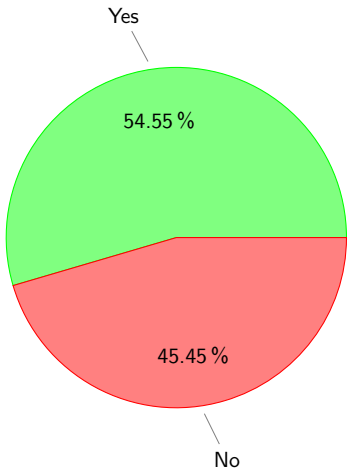


Figure B.28: “Would I have needed more time to do the exercises?”

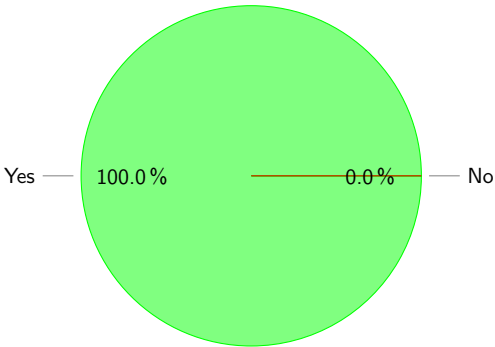


Figure B.29: “Did I understand the principle of sequential execution (execution of the blocks one after the other)?”

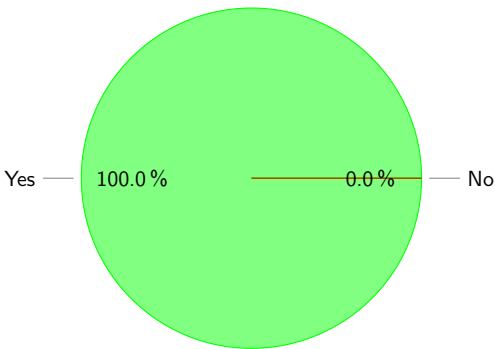


Figure B.30: “Did I understand the principle of conditions? (If ...do ...otherwise)”

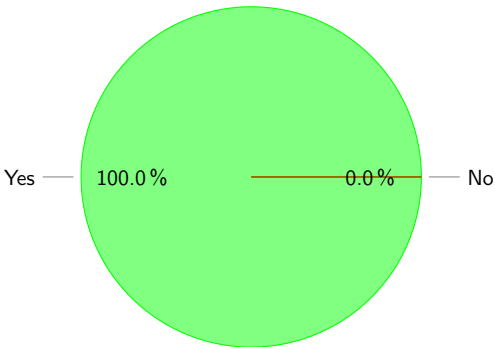


Figure B.31: “Did I understand the principle of loops? (repeat ...)”

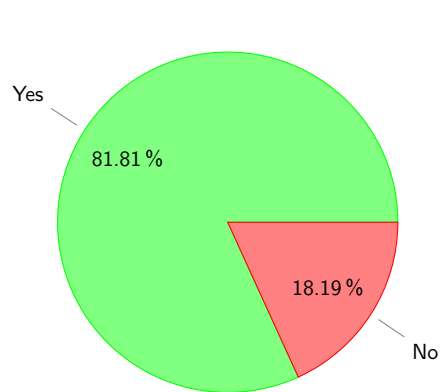


Figure B.32: "Did I understand the principle of functions? (Set of blocks with name)"

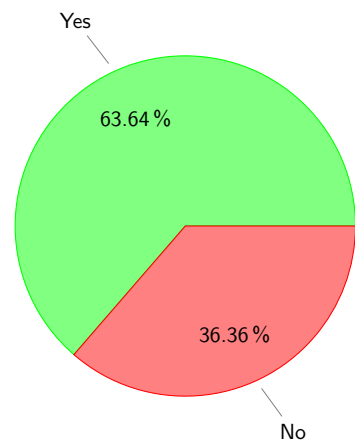


Figure B.33: "Was it easy to use the interface? (Move the blocks, test the code, ...)"

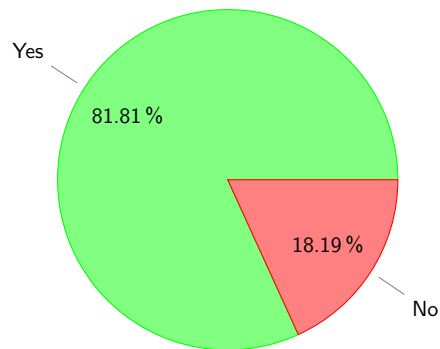


Figure B.34: "Do I want to do other exercises (games to solve with blocks) at home?"

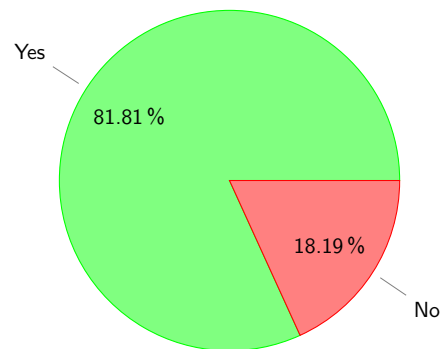


Figure B.35: "Do I receive enough help to be able to understand the tasks?"

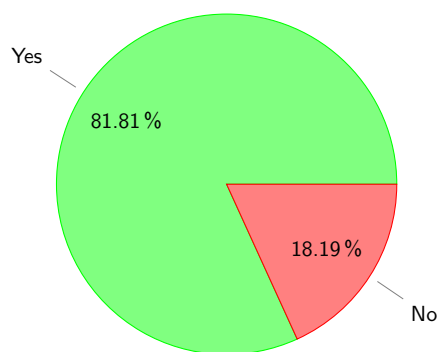


Figure B.36: "Would I like to do this at school?"

B.5 FIFTH ACTIVITY

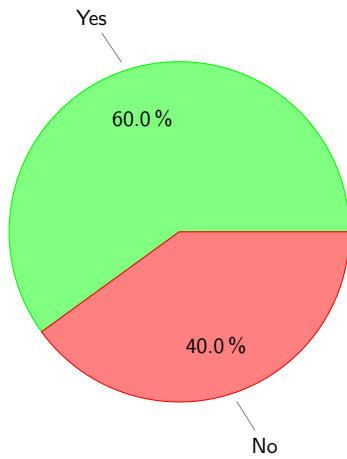


Figure B.37: “Would I have needed more time to do the exercises?”

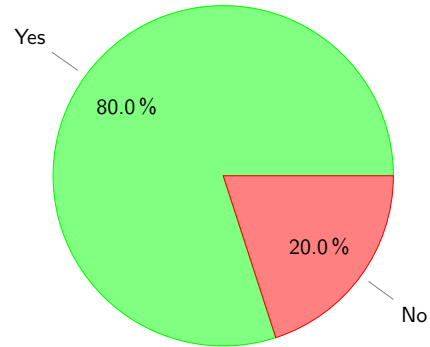


Figure B.38: “Did I understand the principle of sequential execution (execution of the blocks one after the other)?”

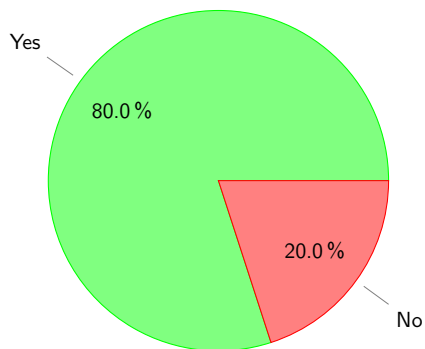


Figure B.39: “Did I understand the principle of conditions? (If ... do ... otherwise)”

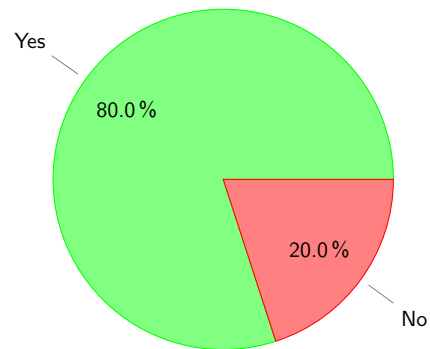


Figure B.40: “Did I understand the principle of loops? (repeat ...)”

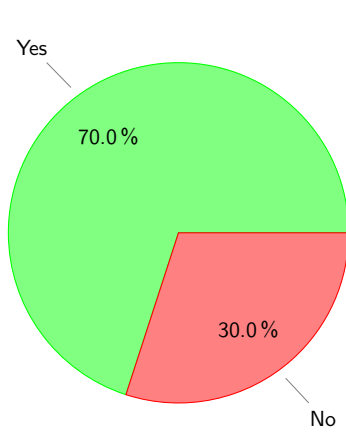


Figure B.41: "Did I understand the principle of functions? (Set of blocks with a name)"

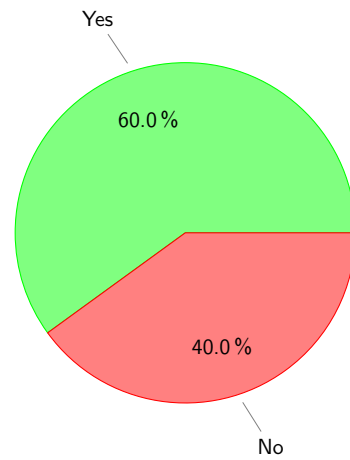


Figure B.42: "Was it easy to use the interface? (Move the blocks, test the code, ...)"

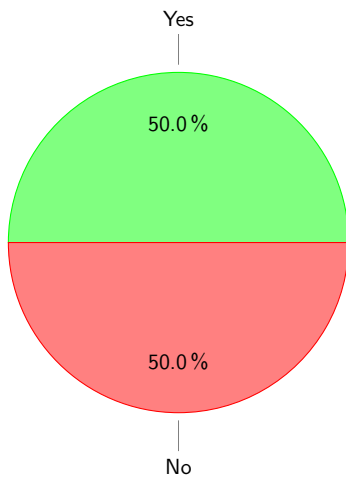


Figure B.43: "Do I want to do other exercises (games to solve with blocks) at home?"

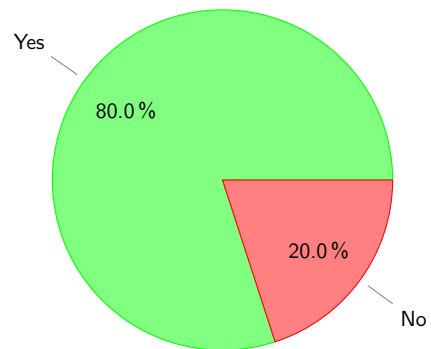


Figure B.44: "Do I receive enough help to be able to understand the tasks?"

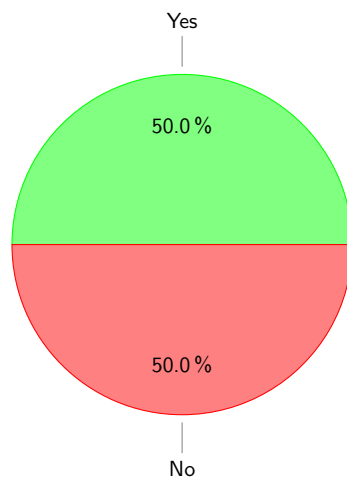


Figure B.45: "Would I like to do this at school?"

B.6 SIXTH ACTIVITY

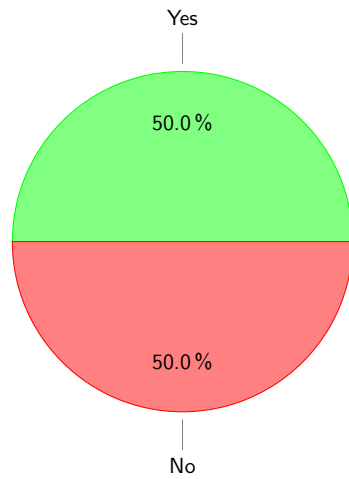


Figure B.46: "Would I have needed more time to do the exercises?"

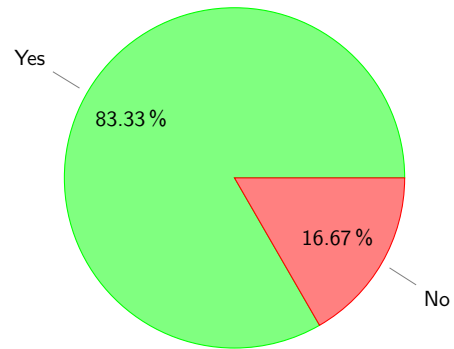


Figure B.47: "Did I understand the principle of sequential execution (execution of the blocks one after the other)?"

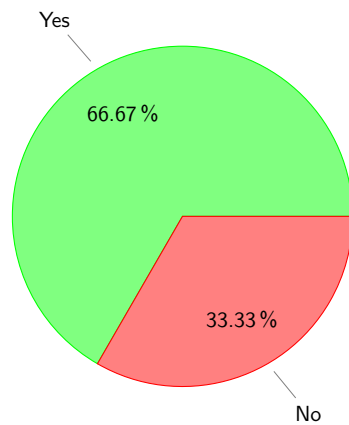


Figure B.48: "Did I understand the principle of conditions? (If ... do ... otherwise)"

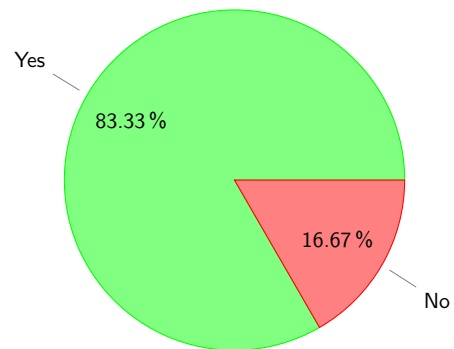


Figure B.49: "Did I understand the principle of loops? (repeat ...)"

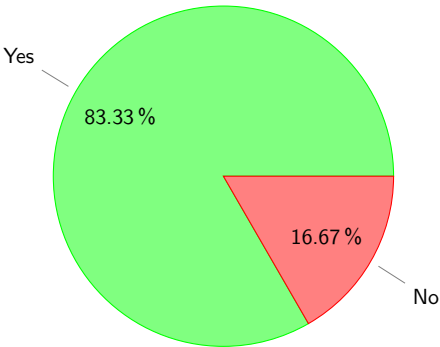


Figure B.50: “Did I understand the principle of functions? (Set of blocks with a name)”

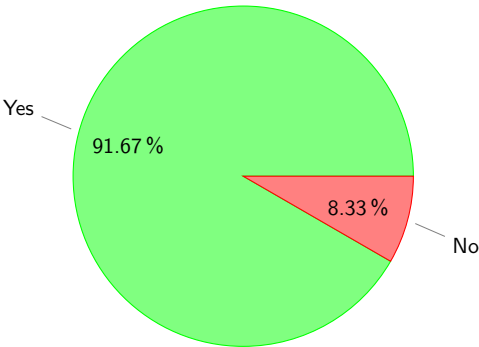


Figure B.51: “Was it easy to use the interface? (Move the blocks, test the code, ...)”

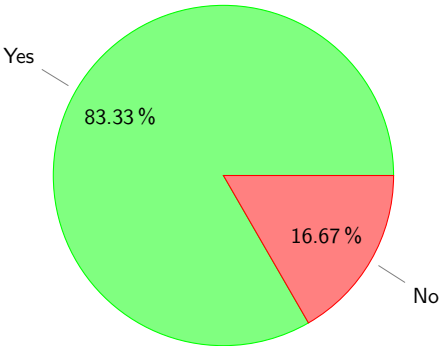


Figure B.52: “Do I want to do other exercises (games to solve with blocks) at home?”

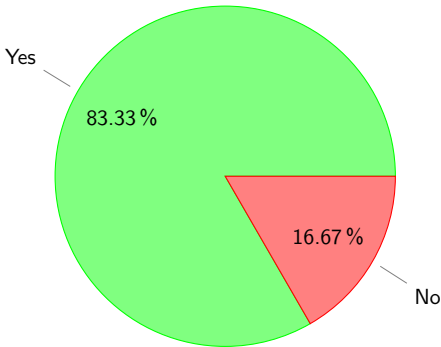


Figure B.53: “Do I receive enough help to be able to understand the tasks?”

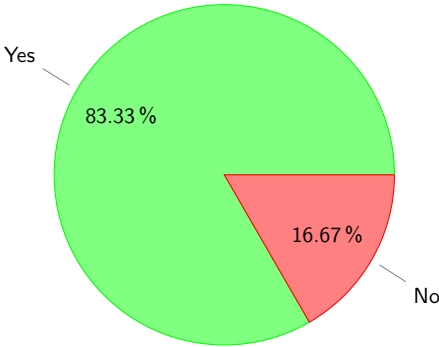


Figure B.54: “Would I like to do this at school?”

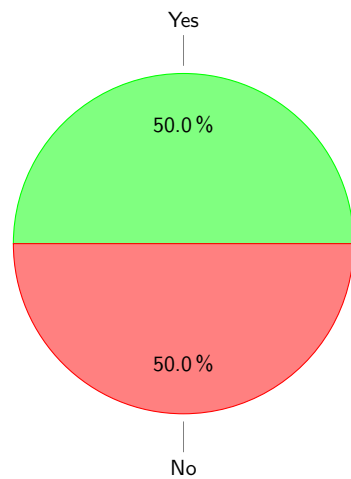


Figure B.55: "Would I have needed more time to do the exercises?"

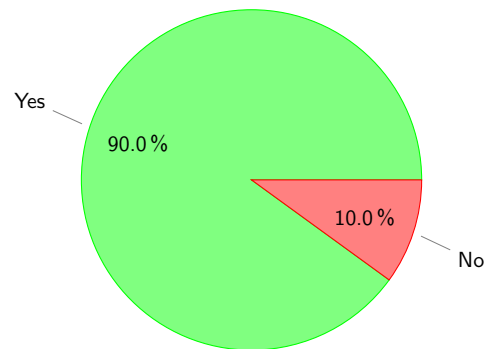


Figure B.56: "Did I understand the principle of sequential execution (execution of the blocks one after the other)?"

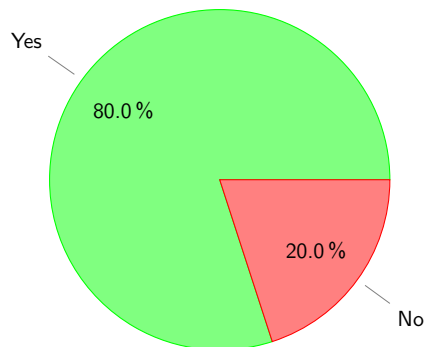


Figure B.57: "Did I understand the principle of conditions? (If ... do ... otherwise)"

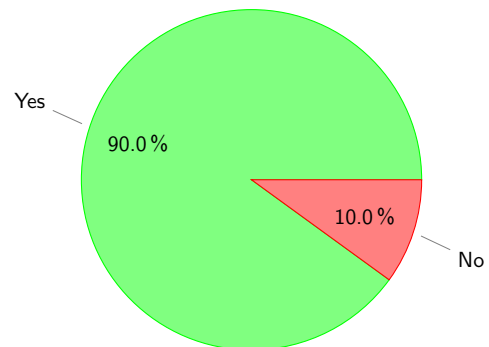


Figure B.58: "Did I understand the principle of loops? (repeat ...)"

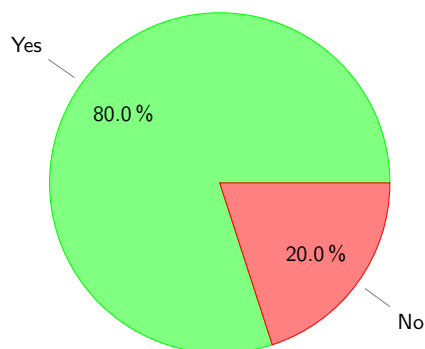


Figure B.59: "Did I understand the principle of functions? (Set of blocks with a name)"

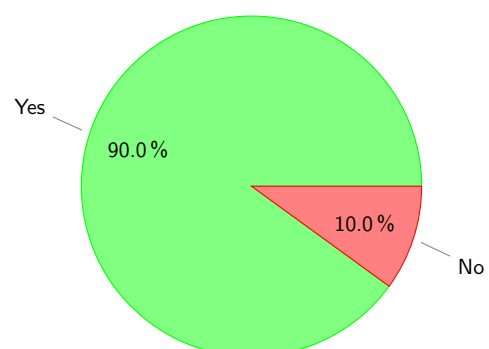


Figure B.60: "Was it easy to use the interface? (Move the blocks, test the code, ...)"

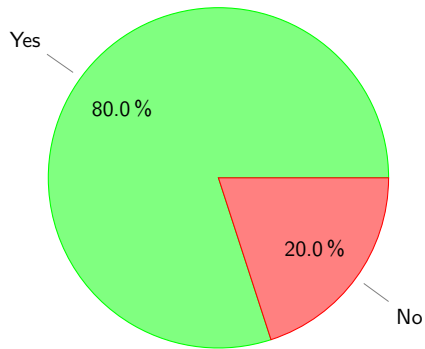


Figure B.61: "Do I want to do other exercises (games to solve with blocks) at home?"

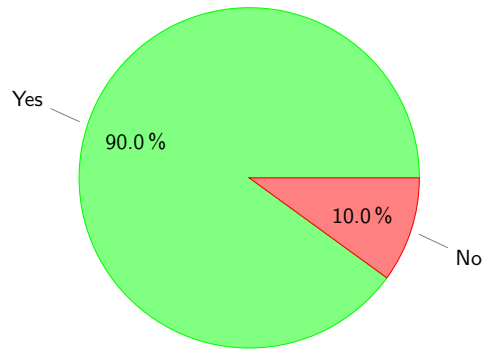


Figure B.62: "Do I receive enough help to be able to understand the tasks?"

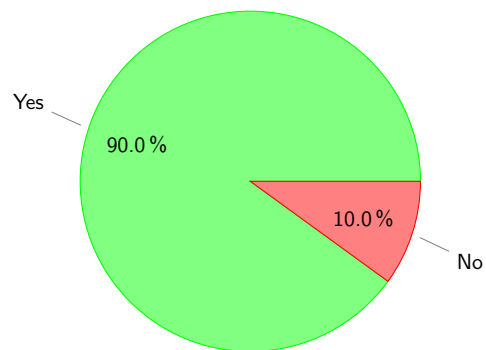


Figure B.63: "Would I like to do this at school?"

LIST OF FIGURES

Figure 1.1	Blockly blocks	4
Figure 2.1	Agentsheets interface	8
Figure 2.2	Etoys interface	8
Figure 2.3	Ardublock interface	9
Figure 2.4	Blockly interface as shown on the official website	10
Figure 2.5	Scratch interface	11
Figure 2.6	Snap interface	12
Figure 3.1	Basic architecture of INGINious	16
Figure 3.2	Architecture of the courses in INGINious	17
Figure 3.3	Blockly interface	18
Figure 3.4	Blockly repeat block	20
Figure 3.5	Blockly print block	20
Figure 3.6	Block repeat and print used together	21
Figure 4.1	UML diagram of the Problem classes	24
Figure 4.2	UML diagram of the Box classes	25
Figure 4.3	UML diagram for DisplayableBlocklyBox class	26
Figure 4.4	An example of Blockly problem displayed in an INGINious task page	28
Figure 4.5	An introduction task from the course Learn Python resolved with Blockly	30
Figure 4.6	The maze activity from the Blockly Games	31
Figure 4.7	The Maze from Blockly Games integrated to INGINious	34
Figure 4.8	The Blockly Factory for an INGINious problem	35
Figure 5.1	Example of Plant VS Zombie problem	39
Figure 5.2	A problem from the <i>Bee</i> course	41
Figure 5.3	Tasks completed over time by each pair of students (March 20, 2017)	42
Figure 5.4	Tasks completed over time by each pair of students (March 21, 2017)	43
Figure 5.5	Tasks completed over time by each pair of students (part 1 March 23, 2017)	44
Figure 5.6	Tasks completed over time by each pair of students (part 2 March 23, 2017)	44
Figure 5.7	Tasks completed over time by each pair of students (part 1 March 24, 2017)	45

Figure 5.8	Tasks completed over time by each pair of students (part 2 March 20, 2017)	46	
Figure 5.9	Tasks completed over time by each pair of students (part 1 March 26, 2017)	47	
Figure 5.10	Tasks completed over time by each pair of students (part 2 March 20, 2017)	48	
Figure 5.11	Plant VS Zombie task 6	51	
Figure 5.12	Solution to task 6 of Plant VS Zombie using simple loops	52	
Figure 5.13	Solution to the task 6 of Plant VS Zombie using imbricated loops	52	
Figure B.1	"Would I have needed more time to do the exercises?"	63	
Figure B.2	"Did I understand the principle of sequential execution (execution of the blocks one after the other)?"	63	
Figure B.3	"Did I understand the principle of conditions? (If ... do ... otherwise)"	63	63
Figure B.4	"Did I understand the principle of loops? (repeat ...)"	63	
Figure B.5	"Did I understand the principle of functions? (Set of blocks with a name)"	64	
Figure B.6	"Was it easy to use the interface? (Move the blocks, test the code, ...)"	64	
Figure B.7	"Do I want to do other exercises (games to solve with blocks) at home?"	64	
Figure B.8	"Do I receive enough help to be able to understand the tasks?"	64	64
Figure B.9	"Would I like to do this at school?"	64	
Figure B.10	"Would I have needed more time to do the exercises?"	65	
Figure B.11	"Did I understand the principle of sequential execution (execution of the blocks one after the other)?"	65	
Figure B.12	"Did I understand the principle of conditions? (If ... do ... otherwise)"	65	65
Figure B.13	"Did I understand the principle of loops? (repeat ...)"	65	
Figure B.14	"Did I understand the principle of functions? (Set of blocks with a name)"	65	
Figure B.15	"Was it easy to use the interface? (Move the blocks, test the code, ...)"	65	
Figure B.16	"Do I want to do other exercises (games to solve with blocks) at home?"	66	
Figure B.17	"Do I receive enough help to be able to understand the tasks?"	66	66
Figure B.18	"Would I like to do this at school?"	66	
Figure B.19	"Would I have needed more time to do the exercises?"	67	
Figure B.20	"Did I understand the principle of sequential execution (execution of the blocks one after the other)?"	67	
Figure B.21	"Did I understand the principle of conditions? (If ... do ... otherwise)"	67	67
Figure B.22	"Did I understand the principle of loops? (repeat ...)"	67	

Figure B.23	"Did I understand the principle of functions? (Set of blocks with a name)"	68	
Figure B.24	"Was it easy to use the interface? (Move the blocks, test the code, ...)"	68	
Figure B.25	"Do I want to do other exercises (games to solve with blocks) at home?"	68	
Figure B.26	"Do I receive enough help to be able to understand the tasks?"		68
Figure B.27	"Would I like to do this at school?"	68	
Figure B.28	"Would I have needed more time to do the exercises?"	69	
Figure B.29	"Did I understand the principle of sequential execution (execution of the blocks one after the other)?"	69	
Figure B.30	"Did I understand the principle of conditions? (If ... do ... otherwise)"		69
Figure B.31	"Did I understand the principle of loops? (repeat ...)"	69	
Figure B.32	"Did I understand the principle of functions? (Set of blocks with a name)"	70	
Figure B.33	"Was it easy to use the interface? (Move the blocks, test the code, ...)"	70	
Figure B.34	"Do I want to do other exercises (games to solve with blocks) at home?"	70	
Figure B.35	"Do I receive enough help to be able to understand the tasks?"		70
Figure B.36	"Would I like to do this at school?"	70	
Figure B.37	"Would I have needed more time to do the exercises?"	71	
Figure B.38	"Did I understand the principle of sequential execution (execution of the blocks one after the other)?"	71	
Figure B.39	"Did I understand the principle of conditions? (If ... do ... otherwise)"		71
Figure B.40	"Did I understand the principle of loops? (repeat ...)"	71	
Figure B.41	"Did I understand the principle of functions? (Set of blocks with a name)"	72	
Figure B.42	"Was it easy to use the interface? (Move the blocks, test the code, ...)"	72	
Figure B.43	"Do I want to do other exercises (games to solve with blocks) at home?"	72	
Figure B.44	"Do I receive enough help to be able to understand the tasks?"		72
Figure B.45	"Would I like to do this at school?"	73	
Figure B.46	"Would I have needed more time to do the exercises?"	74	
Figure B.47	"Did I understand the principle of sequential execution (execution of the blocks one after the other)?"	74	
Figure B.48	"Did I understand the principle of conditions? (If ... do ... otherwise)"		74
Figure B.49	"Did I understand the principle of loops? (repeat ...)"	74	
Figure B.50	"Did I understand the principle of functions? (Set of blocks with a name)"	75	

Figure B.51	“Was it easy to use the interface? (Move the blocks, test the code, ...)”	75
Figure B.52	“Do I want to do other exercises (games to solve with blocks) at home?”	75
Figure B.53	“Do I receive enough help to be able to understand the tasks?”	75
Figure B.54	“Would I like to do this at school?”	75
Figure B.55	“Would I have needed more time to do the exercises?”	76
Figure B.56	“Did I understand the principle of sequential execution (execution of the blocks one after the other)?”	76
Figure B.57	“Did I understand the principle of conditions? (If ... do ... otherwise)”	76
Figure B.58	“Did I understand the principle of loops? (repeat ...)”	76
Figure B.59	“Did I understand the principle of functions? (Set of blocks with a name)”	76
Figure B.60	“Was it easy to use the interface? (Move the blocks, test the code, ...)”	76
Figure B.61	“Do I want to do other exercises (games to solve with blocks) at home?”	77
Figure B.62	“Do I receive enough help to be able to understand the tasks?”	77
Figure B.63	“Would I like to do this at school?”	77

LIST OF TABLES

Table 5.1	Schedule of the activities	38
Table 5.2	Workstation specifications (Sun Ultra 24)	38
Table 5.3	Concepts addressed in each task of the Plant VS Zombie activity	40
Table 5.4	Concepts addressed in each task of the Bee activity	41

LISTINGS

Listing 1.1	Python code equivalence	4
Listing 3.1	Repeat block definition (JSON)	19
Listing 3.2	Print block definition (JavaScript)	19
Listing 3.3	Block repeat code generation for Python	20

Listing 3.4	Block print code generation for Python	20	
Listing 3.5	Translation of the a print block inside of a repeat block in Python		21
Listing 4.1	Definition of the block used to move forward in the maze	32	
Listing 4.2	A function to move the character forward or backward	33	
Listing 4.3	Example of native function added to the interpreter	33	

ACRONYMS

MIT	Massachussetts Institute of Technology
STEM	Science, Technology, Engineering and Mathematics
UCL	Université catholique de Louvain
VPL	Visual Programming Language
IDE	Integrated Development Environment
GUI	Graphical User Interface

BIBLIOGRAPHY

- [1] *Wikipedia, Programming Language*. https://en.wikipedia.org/wiki/Programming_language. Accessed 2017-05-03.
- [2] Maria Hristova, Ananya Misra, Megan Rutter, and Rebecca Mercuri. *Identifying and correcting Java programming errors for introductory computer science students*. Ed. by Scott Grissom, Deborah Knox, Daniel T. Joyce, and Wanda Dann. Feb. 15, 2006. URL: <http://dblp.uni-trier.de/db/conf/sigcse/sigcse2003.html#HristovaMRM03>; <http://doi.acm.org/10.1145/611892.611956>.
- [3] *Github, UCL-INGI/INGInious project*. <https://github.com/UCL-INGI/INGInious>. Accessed 2017-05-03.
- [4] *Peut-on vraiment parler de fossé numérique?* http://www.luttepauvrete.be/chiffres_fosse_numerique.htm. July 2016.
- [5] Hadi Partovi. *16% of all new wages in the U.S. are in computer science*. <http://blog.code.org/post/143085018748/16-of-all-new-wages-in-the-us-are-in-computer>. Apr. 2016.
- [6] *The College Board*. <https://www.collegeboard.org/about>.
- [7] Rick Morgan and John Klaric. *AP Students in College: An Analysis of Five-Year Academic Careers*. <http://research.collegeboard.org/sites/default/files/publications/2012/7/researchreport-2007-4-ap-students-college-analysis-five-year-academic-careers.pdf>. Apr. 2007.
- [8] Académie des sciences. *L'enseignement de l'informatique en France. Il est urgent de ne plus attendre*. 2013.
- [9] *AgentSheets, coding for kids*. <http://www.agentsheets.com/>.
- [10] *Squeakland: home of squeak etoys*. <http://www.squeakland.org/>.
- [11] *Interview with Neil Fraser, Blockly development team*. <https://opensource.com/education/15/2/blockly-makes-easier-every-one-learn-code>.
- [12] *Scratch statistics*. <https://scratch.mit.edu/statistics>.
- [13] *Usage share of web browsers, Wikipedia*. https://en.wikipedia.org/wiki/Usage_share_of_web_browsers.
- [14] *Intent to implement HTML5 by default, Chromium Google Group*. [https://groups.google.com/a/chromium.org/forum/#!searchin/chromium-dev/HTML5\\$20by\\$20default/chromium-dev/0wWoRRhTA_E/_E3jf400AAJ](https://groups.google.com/a/chromium.org/forum/#!searchin/chromium-dev/HTML5$20by$20default/chromium-dev/0wWoRRhTA_E/_E3jf400AAJ).

- [15] Simon Claessens and Gaëtan Collart. “Plateforme web pour accompagner l’apprentissage de la programmation par les 10-14 ans.” MA thesis. Université catholique de Louvain, 2014.
- [16] Simon Hock. “RSnap: Une plateforme web pour accompagner l’apprentissage de la programmation de 10 à 14 ans.” MA thesis. Université catholique de Louvain, 2015.
- [17] *Blockly Developer Tools demo*. <https://blockly-demo.appspot.com/static/demos/blockfactory/index.html>.
- [18] *A sandboxed JavaScript interpreter in JavaScript*. <https://github.com/NeilFraser/JS-Interpreter>.
- [19] *INGInious production instance used at UCL*. <https://inginius.info.ucl.ac.be/index>.
- [20] Marina Papastergiou. “Digital Game-Based Learning in high school Computer Science education: Impact on educational effectiveness and student motivation.” In: *Computers & Education* 52.1 (July 30, 2009), pp. 1–12. URL: <http://dblp.uni-trier.de/db/journals/ce/ce52.html#Papastergiou09a>; <http://dx.doi.org/10.1016/j.compedu.2008.06.004>.
- [21] *Blockly Games demo on App Engine*. <https://blockly-games.appspot.com/>.
- [22] *Code.org website*. <https://code.org/learn>.
- [23] *List of the most downloaded Android applications*. https://en.wikipedia.org/wiki/List_of_most_downloaded_Android_applications.
- [24] *Wikipedia, Plants vs Zombies*. https://en.wikipedia.org/wiki/Plants_vs._Zombies.
- [25] *Discussion about the support of additional languages in INGInious*. https://github.com/UCL-INGI/INGInious/pull/203#discussion_r112393446.
- [26] *INGInious Open Week*. <http://openweek.info.ucl.ac.be/>.
- [27] *BlockPy, a web-based python environment to work with block, text or both*. <http://think.cs.vt.edu/blockpy/>.

