Numeric Line: A Códimo's Activity

Luciano Graziani, Gabriela Anahí Cayú, Matías Emanuel Sanhueza and Enrique Molinari[1],

Universidad Nacional de Río Negro, Sede Atlántica, Viedma, Río Negro
https://www.unrn.edu.ar/
{lgraziani, gcayu, msanhueza, emolinari}@unrn.edu.ar

Abstract. Computational thinking is a fundamental skill for everyone, not just for computer scientists. Teaching computational thinking in primary and secondary school will not only help to inspire kids to continue their studies in Computer Science, but also will provide a set of valuable skills to use during their professional life in the Computer Science field or in any other field. In this work we present a Códimo's activity called Numeric Line. This activity was designed for students between 6 and 8 years old and with the goal of encourage, in a transparent way, one of the aspect of computational thinking: Algorithm design. The students play using a set of simple blocks (commands), trying to put a number in a numeric line, moving it across a maze. With this, the student reinforce their knowledge taught in classroom about natural and integer numbers, using computational thinking.

Keywords: códimo, education, computational-thinking, learning-objects, programming-for-kids, javascript.

1 Introduction

Since 2014 the Universidad Nacional de Río Negro, has been promoting activities to bring programming to primary and secondary students. The goal was to try to inspire the students to continue their studies in computer science. In addition, Computational thinking is a fundamental skill for everyone, not just for computer scientists [9].

After two years of experience doing this, using tools like Alice, and seeing the difficulties in teachers and students we decided to build a tool to reinforce concepts taught in classroom, but using computational thinking. We called this tool Códimo.

In all Códimo's activities, the students reinforce concepts learnt in classroom, in any field like exact science, natural science, social science or any other, using concepts from computational thinking.

This work follows with section 2, with the description of our proposal and a detailed explanation of the activity Numeric Line. The section 3 describing technical details of how Códimo was built. The section 4 with related work, the section 5 with future work to finish with section 6, the conclusions.

[1] Corresponding author.

2 “Numeric Line”: Coding to Learn

Numeric line is an activity for students from 6 to 8 years old (2nd grade and 3rd grade). It belongs to math area and can be used for students after their learn natural numbers, integers, the order relation and the numeric line.

The goal of the activity is to put a number in an empty position in the numeric line. It presents several levels of complexity. In the initial level the students will play with natural numbers of one digit. In the intermediate level with natural numbers of two digits and in the advanced level the students will play with integer numbers.

The activity presents a number in an initial position, a maze and a numeric line. The students must move the number and put it into the numeric line in a correct position. The only way to move the number is using blocks. Each block has a specific purpose like move the number in a direction or make the number to jump. To take the number from the initial position to an end position in the numeric line, the students must create a *program* joining the given blocks. Playing like this, the students reinforce math applying algorithm design, one of the aspects of computational thinking.

Using blocks to code has a fundamental pedagogic advantage over using classic syntactical constructions from a pseudo programming language. The blocks always form a syntactically correct *program*, which allows the students to don’t have to deal with semicolons or non-closing brackets, and put the attention on the semantics, the effect that each block produce to the number. In this case, the effect is to move the number across the maze.

Figure 1 shows one of the exercises from the advanced level.

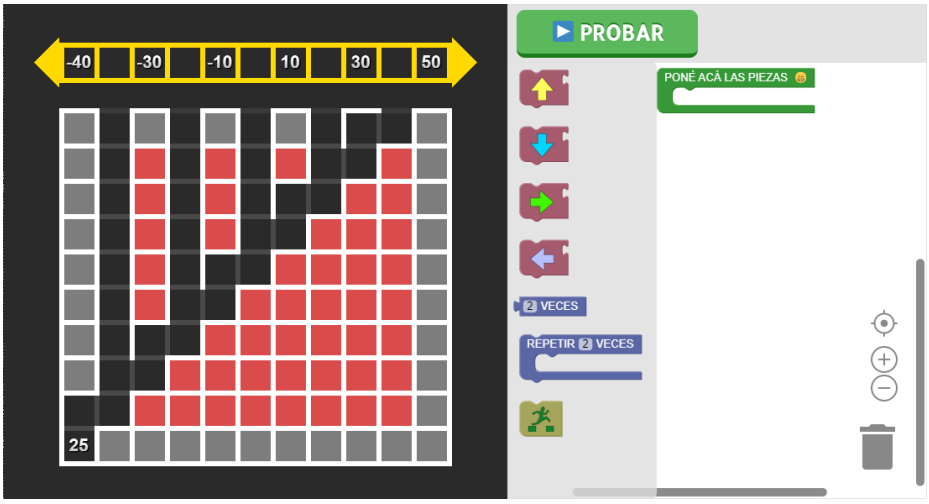

Fig. 1. Activity, advanced level

On the left of Fig. 1 you can see the maze and on top of it a numeric line. Each activity start with a number, in the case of the Fig. 1 the 25, and a numeric line with some empty positions. The number 25 must be put in an empty correct position in the numeric line. In the initial level the numeric line has only one empty position, but in

the other levels there are more than one empty position, where one position is valid to put the number.

On the right of Fig. 1 you can see the blocks used by the students to move the number. Each block produce a defined effect on the number. We classify the blocks in the following way:

- Simple:
  - **Move up**: move the number up one position.
  - **Move to right**: move the number to the right one position.
  - **Move to left**: move the number to the left one position.
  - **Move down**: move the number down one position.
- Composed:
  - **N times**: repeat a single move in any direction n times.
  - **Repeat N times**: execute N times the set of simple blocks inside it.
- Designed for the activity:
  - **Jump**: once the number arrives to a valid position, the students must use this block to jump from the maze into the number line.

The blocks must be put inside the green container with a description "*Poné las piezas acá adentro*" (Put the pieces here.)

With the *program* created the button "*Probar*" (Try it) can be pressed, there is when the program is interpreted moving the number across the maze. Once the execution finish, if the number ends up in a valid position, the activity propose to go to the next activity (in the same or in the next level) otherwise the activity can be repeat it. If the activity is repeated, the number to be moved change.

It is important for the authors to emphasize that "Numeric Line" is a math activity not a programming activity. Using the blocks we are stimulating one of the aspect of the computational thinking: algorithm design. But this is not the main goal of the activity which is to reinforce math (natural numbers, integers and the order in the numeric line). All the activities generated with Códimo are designed around this main goal because we think that will make teachers to use them during classroom without needing to deviate from their plan.

We say that the students will "code to learn", due to the main goal is to reinforce a specific topic given by their teachers instead of doing a programming activity where on the other hand, we would call "learn to code".

3 "Numeric Line": Technical Description

Códimo is a web application written in Javascript only. It works in the latest versions of the major browsers and, since its development is open source, in addition to see the code, is possible to collaborate by reporting errors, proposing ideas, or even making modifications. The repository can be found at: https://github.com/lgraziani2712/codimo.

3.1 Technologies used

To be able to develop Códimo was necessary a wide set of tools, both for improve the Development Experience and for assure a good component design. Here below are listed the more meaningful tools and their purpose during development.

Application dependencies

- **Blockly**: "A library for building visual programming editors" [4]. It allows to implement a programming interface using blocks.
- **PixiJS**: "The [...] 2D WebGL renderer" [6]. Its role is to be the 2D Graphic Engine for the activities, that is to say, every animation for the "Numeric Line" activity are developed with this tool.
- **React**: "A Javascript library for building user interfaces" [3]. Is used for the construction of the web page, with the capability of creating dynamic routes without the need of developing a server. It allowed to integrate PixiJS and Blockly without much effort.

In the following figure can be appreciated how was designed the architecture of Códimo:

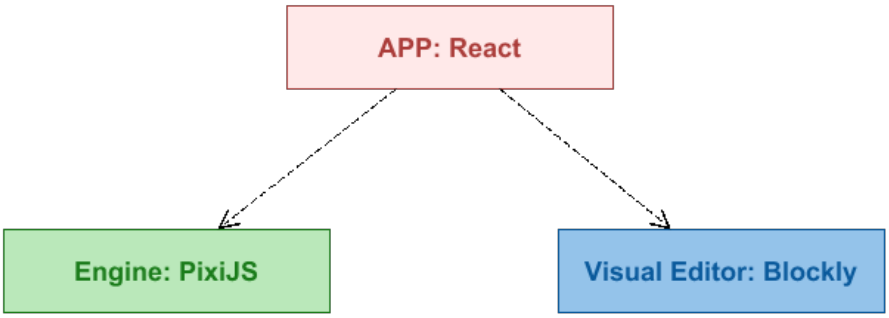

Fig. 2. Códimo Architecture

As the Fig. 2 evidence, the architecture is integrated by three main components:

- **APP**: is responsible for manipulating the application URI's, as the load & unload of every dependency related to specific activities.
- **Engine**: is responsible for rendering the activity, parsing the sequence of instructions made by the student through the **Visual Editor**, and animating every single object according to the instructions that parse. Expose an interface used by the **APP**.
- **Visual Editor**: is responsible for listing every possible instruction for each activity, and offering a place for the student to create the solution using those instructions. Expose an interface used by the **APP**.

Each time the *APP* tries to load an activity, adds the *Engine* instance to the main renderer and request to generate the initial state of the activity. At the same time, adds the *Visual Editor*'s instance and request to create an editor with the necessary instructions for that specific activity.

The student always interacts with the *APP* and this delegates the tasks to the other two components.

Development Experience dependencies

- **Storybook**: "The UI Development Environment You'll ♥ to use" [8]. With this tool was possible to design and develop each visual component individually.
  To Códimo, a visual component not only refers to the text and buttons that go with the activities, but the *Engine* itself. Thus, was possible to design, for example, each animation of the number in an isolated manner.
- **Babel**: allows to transpile the latest versions of the Javascript language to code that can run in the latest versions of the major browsers. With this technology is possible to confirm that Códimo works at least with the latest two versions of Firefox, Chrome, Edge and Safari.
- **ESLint**: statically evaluates the Javascript code in search of syntax, structure and style errors.
- **FlowType**: "is a static type checker for Javascript" [2].
- **Jest**: is a testing framework for Javascript that innovates in certain aspects:
  - It analyses which files were modified since the last commit (when using a GIT repository) and executes by default only the tests related to those files.
  - When using the watcher option (useful while developing), is possible to filter tests from specific files, test names or by a regular expression.
  - **Snapshots**: it's a new way of testing data and raw components. It takes a variable and serialize its content into a file. Each time the test runs, it **diffs** the content of the file with the variable, and if there is a difference, the test fails and reports what was that difference. This is extremely useful when doing unit testing of HTML components since avoids validate each element specifically.
- **Webpack**: is a *module bundler*, a tool that allows to define which file types can be treated as modules. Webpack in particular can consider a wide range of file types (from images, json, css, videos) as a Javascript module, as long as it's configured to do that.
  Also offers a simple, and easy to configure, development server that watches for modifications and hot reload the changes by itself.
  The production version of Códimo, that asynchronous download the files requested by the selected activity and the specific exercise, is generated with Webpack.

Continuous Integration dependencies

- **CircleCI**: in Códimo is used for executing the battery of tests in an isolated environment, and reports if everything passes or not and which tests have failed. It runs every time anyone makes a commit, a Pull Request or a merge to the project.

- **Netlify**: adds a process of continuous deployment to the project. It runs every time anyone makes a commit, a Pull Request or a merge to the project. Also allows to see any version of the website at any time.

4 Related Work

There are various applications in the area that it intends to include Códimo. Some, like *CodeCombat* or *Pilas Engine*, are applications for play or create games or animations (respectively), in which is necessary to code to be able to make actions like moving a character to a specific place, picking an object, o to attack a creature. Another ones make the use of blocks, like *Alice*, *Scratch* or *Pilas Bloques* and others offers free access to educational resources in Computer Science and other disciplines like *Khan Academy*.

In addition to this applications, there is the *Hour of Code*, a global movement that offers a one-hour introductory activities to Computer Science, "designed to demystify "code", to show that anybody can learn the basics, and to broaden participation in the field of computer science" (The Hour of Code, 2017). The event takes place each year during Computer Science Education Week.

A really important element of this event is its repository of activities. Every person and organization can register an activity, among which are from Khan Academy, CodeCombat, Alice and Scratch.

The principal difference between the Hour of Code and Códimo, although subtle, it's significant: the first has as an objective "to show that anybody can learn the basics, and to broaden participation in the field of computer science", not the Computational Thinking per se. Of course, introducing into Programming also introduces into Computational Thinking, however isn't necessary the other way around. On the other hand, it would be highly possible to register an activity to the Hour of Code using Códimo since it was already used for the *Science and Technology Week*, in Argentina on September 2017.

Pilas Engine, on its part, is an Argentine application for develop video games and animations in 2D. Its main advantage is that each component and its methods are in Spanish, so it's possible to do things like this:

```
moneda = pilas.actores.Moneda()
```

However, it's not a Spanish pseudo language. The library *pilas* is written in Python, hence the code that anyone needs to write inevitable mixes English (when writing reserved words) with Spanish (when using the pilas engine library):

```
import pilasengine

class AceitunaEnemiga(pilasengine.actores.Aceituna)
def iniciar(self)
  self.imagen = "aceituna.png"
```

This adds an entry barrier for its use, since a K-12 student has low or zero knowledge of English (written and spoken), and since also doesn't know about programming, it will feel disoriented and it will lost interest quickly. Like CodeCombat, the fact that the student needs to explicitly write code makes this kind of applications unsuitable for introducing Computational Thinking.

Apart from that, exist more friendly applications for whom have their first contact with programming, like Alice: a platform for develop 3D video games and animations. Contains a battery of libraries with object models, characters and scenarios; allows to manipulate the light and camera; and to write a "script". This "script" is written using blocks visually similar to those of the puzzle, preventing to insert them in an unwanted manner. This way, by inserting the proposed blocks, and without explicitly knowing, the student generates an algorithm that it's translated into a game or an animation.

The main difference of Alice in comparison to CodeCombat is the use of blocks as a mechanism to write algorithms, which has a pedagogical fundament and advantage over the use of a programming language: a block always generates a syntactically well written algorithm, allowing students to focus only in understanding the semantic, in other words, the effect produced by the block over other elements. Also, by abstracting the use of the syntax, it allows a completely translation of every instruction.

However, as well as the other applications when compared to Códimo's activities, Alice lacks an essential element: it cannot be used outside the computer class. When Computational Thinking is discussed, is referenced the possibility to apply it in fields outside Computer Science, and it isn't possible to use Alice for that purpose.

As Alice, *Pilas Bloques* uses blocks as a mechanism for writing algorithms. Uses *Pilas Engine* as an engine, is web, and unlike Alice, who offers an empty scene as a canvas, Pilas Bloques presents a set of challenges.

Pilas Bloques is the more similar application to Códimo, since a challenge can represent a simple activity with a specific end: teach iteration, abstraction, modularization or conditionals. Anyway, it lacks of challenges that adapts activities from other disciplines.

5 Future Work

The authors will continue this job of introducing the computational thinking in the primary and secondary school with the following activities:

- Incorporate into the teaching process an evaluation. Applying FLOM [7] as the basis for activities design.
- Evolve Códimo to a framework, with clear extension points, in order to reduce the effort and time to create new activities. Any Javascript developer should be able to use Códimo to create activities.
- The computational thinking has several aspects, not only algorithm design, which was the one used in for activity. The abstraction, pattern recognition, the decomposition of a problem into multiple simple problems are others important aspects. It is our goal create more activities with this idea of "Code to learn" but applying also these other aspects of the computational thinking.

6 Conclusion

In this work we presented "Numeric Line", an activity implemented with Códimo. The goal of this activity is to introduce computational thinking in the primary school, in a way that result attractive for teachers and students. For the teachers because they don't need to deviate from their subjects and incorporating an activity with this characteristics gives them a new tool to motivate students to study. For the students, because they will acquire computation thinking skills while playing reinforcing concepts taught in classroom.

The main distinction from our proposal is the way in which we introduce computational skills. It is a transparent way. The goal is always to focus and work in the concepts related to the subjects from the curriculum, but to work in a different way with a new tool that encourage students to apply computational thinking.

References

1. Códimo. (2017) *¡Donde el código, la imaginación y la motivación se encuentran!* Retrieved June, 2017, from https://codimo.netlify.com/.
2. Facebook. (2017). *Flow is a static type checker for javascript*. Retrieved June, 2017, from https://flow.org/.
3. Facebook. (2017). *React: A Javascript Library for Building User Interfaces*. Retrieved June, 2017, from https://facebook.github.io/react/.
4. Google. (2017). *A library for building visual programming editors*. Retrieved June, 2017, from https://developers.google.com/blockly/.
5. Graziani, L. et al. (2016). *Códimo: Desarrollo del pensamiento computacional en las escuelas*. II Jornadas Argentinas De Tecnología, Innovación y Creatividad 2016 - Workshop sobre Innovación en Centros Educativos y de Investigación (WICEI). Retrieved June, 2017, from http://jatic2016.ucaecemdp.edu.ar/trabajos/WICEI394TE-GrazianiJATIC2016.pdf.
6. PixiJS. (2017). *The HTML5 Creation Engine*. Retrieved June, 2017, from http://www.pixijs.com/.
7. Sanchez, A. J., Perez-Lezama, C. and Starostenko O. (2015). *A Formal Specification for the Collaborative Development of Learning Objects*. Procedia - Social and Behavioral Sciences. 182, 726–731. Retrieved June, 2017, from http://www.sciencedirect.com/science/article/pii/S1877042815030955.
8. Storybook. (2017). *Storybook: The UI Development Environment You'll ♥ to use*. Retrieved June, 2017, from https://storybook.js.org/.
9. Wing, J. M. (2006). *Computational thinking*. Communications of the ACM, 49:33–35. Retrieved June, 2017, from https://goo.gl/CHS4Yp.