Olympiads in Informatics, 2016, Vol. 10, 3–18
© 2016 *IOI, Vilnius University*
DOI: 10.15388/ioi.2016.01

eSeeCode: Creating a Computer Language from Teaching Experiences

Joan ALEMANY FLOS[1], Jacobo VILELLA VILAHUR[2]
[1]*Fundació Aula: 34 Av. Mare de Déu de Lorda, 08034 Barcelona Spain*
Explorium: Samalús 1 L3, 08530 La Garriga Spain
eSeeCode.com
[2]*Aula Escola Europea: 34 Av. Mare de Déu de Lorda, 08034 Barcelona Spain*
eSeeCode.com
e-mail: jalemany@eseecode.com, jvilella@eseecode.com

Abstract. It has been almost 50 years since the creation of Logo – one of the first educational programming languages. Although most people agreed that it was important to teach computational thinking to children, only recently have school and district leaders begun to adopt curricula to include it – mainly through Scratch. In many cases this adoption has failed to provide the right methodologies and tools to teachers.

In this paper, we analyse some of the different languages used in classrooms today and we propose an improved alternative that we have created – eSeeCode. We also share our experiences using this language in classrooms and how students can learn using this tool.

Keywords: informatics curriculum, promoting informatics, programming language.

1. Overview

Reading Papert's *Mindstorms: Children, Computers, and great ideas,* one can have the feeling that we have not advanced much in the past 35 years (Papert, 1980). Many countries are trying to include Coding as a required skill to learn in schools, either as a specific subject or as part of a technology course. However, in many schools, teachers do not have the resources, materials and/or knowledge to bring computer science and coding into the classroom. This is the case of Spain, among other countries, where computer science has been introduced into the curriculum, but has failed to provide the details on how to implement it properly, thus providing teachers the freedom and responsibility to decide how to teach some basic computer science concepts (Saez-Lopez *et al.*, 2016, Ackovska *et al.,* 2015, Duke *et al.*, 2000).

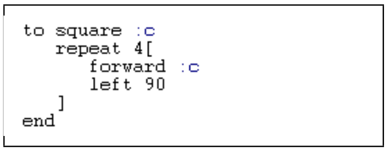
In this paper, we will analyse several computer languages and materials, and we will explain the difficulties students find when using them in class. As Edsger Dijkstra explains, the selection of the programming language will influence how the students will understand computer science (Dijkstra, 1999). We will base our examples on some of the most popular computer languages used in Spain today, although this experience can be extrapolated to many other countries.

1.1. *Key Terms*

To begin, we will use the term “educational computer language” in a broad sense, as a programming language used to teach coding in classrooms, ranging from *professional* computer languages such as **C++** to puzzle-related languages as **Lightbot**. Because in many cases language and programming environment cannot be analysed separately, we will use them indistinguishably.

We will use the term **text-based coding** to describe languages that permit students to type their own code, giving them total freedom in the expressions they write. In contrast, we will use the term **visual block based programming** as the type of coding where you select your own blocks, and build the programs with a drag-and-drop interface. There is a fundamental difference when discussing the use of these languages in the classroom, as the former, generally speaking, cannot prevent students from making syntax errors, while the latter prevents these type of mistakes. Fig. 1 and Fig. 2 show different examples of text-based coding and visual block based programming.

We will differentiate between two kinds of paradigms when discussing different approaches to teaching. The first will be **Problem Solving,** where the teacher can present the student a short, self-contained problem that needs to be answered. In the particular case of this paradigm, we will also use the term **Puzzle Solving.** Similar in concept to problem solving, we consider puzzle solving to have more of a recreational focus, where there is a known set of rules that include multiple variations. For example, when solving a sudoku puzzle there is a specific set of rules, and by changing the numbers



```
to square :c
   repeat 4[
      forward :c
      left 90
   ]
end
```

Fig. 1. Text-based coding example with Logo to create a square.

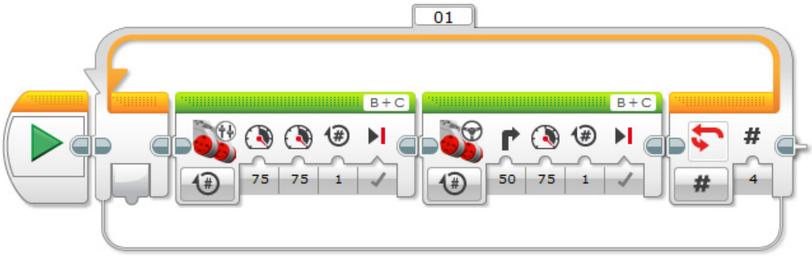


Fig. 2. Visual coding example with EV3 to make a Lego Robot move in a square pattern.

you change the problem. The teacher acts as a problem-setter or planifier, deciding which problems to be solved at each moment, such as when a student is asked to code the Eratosthenes sieve.

As an alternative to problem solving we find **Project-based learning**. In Spain, this strategy is gaining increased attention in the school setting. As part of project-based learning, students take a more active and creative role by pursuing an authentic and real-world project that addresses essential questions and works towards gaining an enduring understanding of a relevant issue. The teacher here is more of a guide or facilitator that helps the student when needed. In computer science this has the form of *software design*, *applied programming* and *modelling*. For example, a student could work on solving the problem of being able to identify a number as prime or not, with the project title being "Create a game that uses prime numbers".

1.2. *Languages*

To help keep the explanation focused, we have created a list of the main languages used in schools in Spain, and have classified them by similarity of characteristics.

The final result is shown below:

Under *text-based coding* we will find two separate groups: the pure educational languages such as Logo or Processing, and the professional languages such as C++, Java, Python and Javascript.

Under *visual block based coding* we find again two groups: educational languages such as Scratch, Alice, Kodu, Ev3, AppInventor, and the group of puzzle languages such as Lightbot and Beebot. Strictly speaking, Lightbot and Beebot are not complete languages, but they are used to teach basic structures to students. It is for this reason that we have decided to include them in our analysis.

Although each language has its own particular characteristic, we will analyse a representative of each group. The languages we have chosen are: Logo, C++, Scratch and Lightbot.

2. Main Characteristics

To be able to compare the different languages we have created a short introduction for each one.

2.1. *Logo*

Originally created by Wally Feurzeig and Seymour Papert in 1967, there are many versions of the language used in schools. Its main objective was to teach programming to students from ages five to thirteen (Papert, 1980). Due to the fact that it has been around for so long, there has been more than 300 versions of the language, with a substantial amount of literature related to it. (Boytchev, 2013).

Main characteristics:

- Educational (ages 5–13).
- The majority of versions use text-code.
- The first activities proposed were problem-solving oriented, although there was some space for creativity. This will vary with versions.
- Uses a drawing area where you can move a pointer called a "Turtle".
- Main academic fields are basic algorithmics and 2D–3D geometry depending on the version.
- In general there were no debuggers, but a step-by-step execution was possible in many versions, which helped locate errors. Also syntax errors are difficult to correct.

2.2. *C++*

Designed by Bjarne Stroustrup in 1983, C++ is a *professional* computer language utilized heavily in many academic and professional circles. It is a compiled language, allowing the user to be able to select a programming environment. Its standard library makes it easier to use, as compared to its predecessor C (Stroustrup, 2007). It is important to note that in this category that there are very different approaches depending on the paradigms used (Duke *et al.*, 2000), but we will not take them into consideration for the current analysis and will common ground.

Main characteristics:

- Professional (ages 12+)
- C++ uses formal code.
- The language accepts both problem-solving activities and project-oriented learning.
- No drawingarea. General interface uses a console to write and read (input/output). This can be extended to use of files.
- Main academic fields are advanced algorithms, data structures, and numerical problems. Uses of classes helps teach system design.
- Depending on the Interface there is a good debugger, but syntax errors are hard to read.

2.3. *Scratch*

Scratch was created in 2005 by Lifelong Kindergarten research group at Massachusetts Institute of Technology Media Lab led by Mitchel Resnick. Although you can trace some of its origins to Logo, its different approach to teaching computer science places it in a different category (Resnick *et al.*, 2009).

Main characteristics:

- Educational (ages 8–16).
- It uses visual block based programming.
- The language is more focused on project-oriented programming.
- There is a "drawing area" used to place objects and move them around.
- Main academic fields cover basic algorithm, software design.
- Due to the visual programming interface with blocks, it is impossible to get syntax errors. There is no debugger.

2.4. *Lightbot*

There is an educational movement promoted by *code.org* to reach children in different regions of the world and give them tutorials to learn how to code. This movement started as the "Hour of code" and has had a big impact all around the world. Almost 200,000 events were carried out in more than 140 countries. (Code.org, 2013–2016)

Many schools around the world use this material as an introductory material for Computer Science. Among the different tutorials that you can find, we selected one that had a large acceptance rate in the teacher community in Spain – Lightbot.

Lightbot is a puzzle-based game where you need to light some squares on a grid. It was created in 2008 by Danny Yaroslavski, but it was with the hour of code that it became popular. (Gouws *et al.*, 2014)

Main characteristics:

- Educational (ages 4–8 and 9+).
- It uses visual block based programming.
- The language is focused only on puzzle solving.
- It has nice animations of a robot moving around.
- Main academic fields cover basic algorithms (no variables).
- Impossible to get syntax errors, the execution can be considered *step-by-step.*

3. Developing a Curriculum

It is clear that these languages are difficult to compare to one another because of their fundamental differences. To be able to do so we will analyse its uses in the classroom and their main drawbacks if used alone. Later we will study the combination of one or more language.

When considering the development of a curriculum we have to take into account many factors, but mainly the maturity of the students (their age) and the amount of time they will spend engaging with that curriculum. In our case we will consider students from primary – secondary schools, and a relatively long length of time of engagement (3–4 years). Although some consider this length insufficient (Winslow, 1996), it is a valid starting point to achieve competence in programming. Some main objectives can be found in some studies (Duncan and Bell, 2015) and in some publications like the Computing Progression Pathways (Dorling, 2014).

3.1. *Teaching with Only One Language*

Using the *code.org* tutorials in the classroom may seem like a good idea because of the students' high level of initial engagement; however, over time this strategy can backfire if it is assigned for too long as the teacher has limited control over the content (s/he cannot put his own problems), and the tutorials are short and very specific.

One of the program's strong points is that the students get positive feedback and do not feel frustrated when they fail. This is probably due to the fact that they view Lightbot and other tutorials in *code.org* as a game, instead of a class problem.

If we look at the drawbacks from Lightbot we can see that the number of commands is very limited and do not include variables. For younger children this is good because the less options you give them the more focused and easy it will be to arrive at solutions. On the other hand it is not good if the students are mature enough to learn and understand it quickly. The use of loops is another drawback. To create a loop one must make a procedure that calls upon itself (as in recursion). This makes an infinite type of loop. Although the students can find it intuitive, they have a difficult time understanding conditionals and being able to predict this kind of behaviour. Fig. 3 shows the use of loops and functions. We have to keep in mind that although some teachers might use it as a tool to teach programming, it does not cover some basic algorithmic concepts that are important, and its programs cannot be generalised. (Lightbot 2016, Gouws *et al.*, 2014)

Currently in Spain the most popular educational programming language is Scratch, where it has become quite popular, evidenced by the fact that Barcelona hosted the Connecting Worlds Scratch Conference in 2013. With Scratch, students are engaged and motivated (Saez-Lopez, 2016), but after many years of use their interest wanes. This is a big drawback because it generates apathy towards programming and, in some cases, a dislike for programming altogether. There are also some technical drawbacks. For example, the lack of text coding makes it difficult to read and write long conditionals and programs. There are some projects to overcome this difficulty (Harvey and Mönig

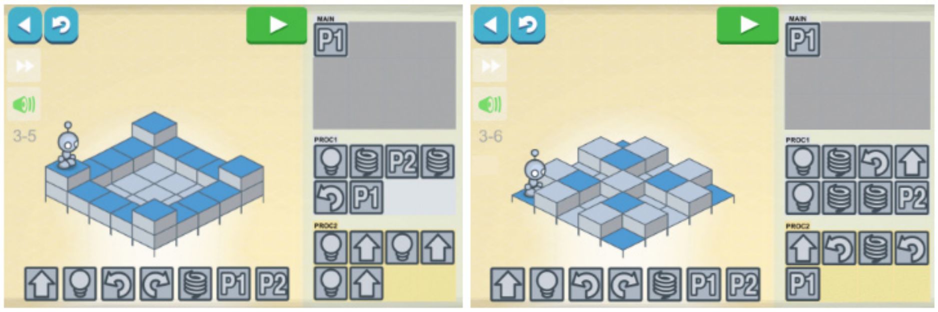
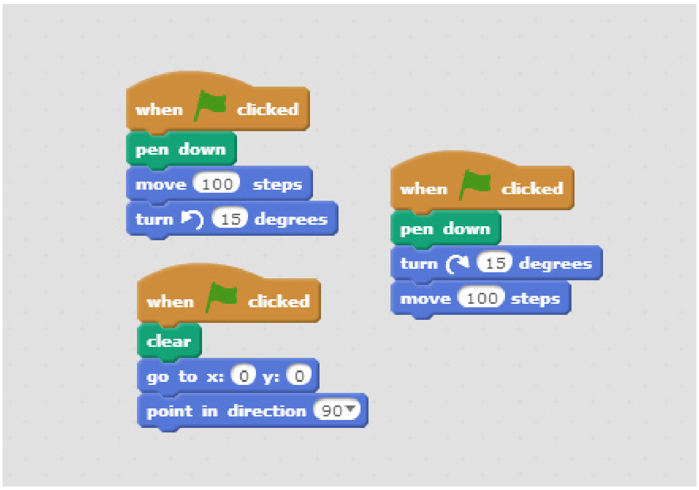


Fig. 3. Lightbot usage of recursive procedures instead of loops.

2010), but it is not included in the main tool. In addition, the high level language makes the student miss the opportunity to understand what is behind some of the instructions. Finally, as it can be seen in Fig. 4, Scratch allows some "parallel programming", which is actually not accurate as it is executed sequentially and depends on the order the procedures were created. This is a problem because the student cannot guess the results of a given program, which is one of the main objectives when teaching CS. (Dorling, 2014)

For a long time the most popular educational programming language in Spain was Logo. It was used in the 1980's through early 1990's, but its use faded away in the late 1990's. We can see this, for example, because it disappeared from the teacher majors in universities. There was not an immediate substitution by another language, but the government stopped promoting it as it was not news. This is reflected, for instance, in the use of it when training teachers at that time. (Simon, 1996) At the onset, students were motivated by this new approach to teaching and learning (Rubinstein, 1974), but today the look of a majority of the Logo platforms appear outdated and need a visual update. There are two major drawbacks to Logo. Its theoretical use was for students aged 5–13, with newer versions this could be extended until age 15, but text-based coding may bring syntax errors, and students need some maturity to be able to correct them. If not well attended students would get frustrated with programs that could not run. At the same time, similarly to Scratch, it has a limited life span in the student studies due to the apparent lack of practical utility.

The last group of languages to analyse is the professional ones. This are the most common choice among high school students. The students see the real value of coding and can explore other areas such as physics and mathematics, but because it is not prepared for the classroom the learning curve is very steep. Winslow describes the five steps from novice to expert (Winslow, 1996), in small-to-medium classrooms, the steep learning curve produces a clear and early separation between those who understand the content (and move through Winslow learning steps) and those who do not and remain as Novices. Another drawback is that due to the lack of visual assistance (in general) and



```
when clicked
pen down
move 100 steps
turn ↺ 15 degrees

when clicked
clear
go to x: 0 y: 0
point in direction 90

when clicked
pen down
turn ↻ 15 degrees
move 100 steps
```

Fig. 4. *Scratch multiple parallel procedures.*

less attractive interface it is more difficult to motivate the students to use it. The syntax of some functions can also be a problem for some students. For instance, the syntax of the *for* loop is more complicated than in other languages, which makes basic algorithms sometimes hard to teach (Finkel *et al.,* 1994, Robins and Rountree, 2003). Depending on the programming environment used, the syntax errors are very difficult to read to the untrained eye, as can be seen in Fig. 5, where only a single character missing produces 20 lines of errors when compiling.

One big advantage is the amount of resources that can be found on the internet, in particular online judges like Codeforces, Timus, Topcoder, etc. With this sort of information it is really easy to prepare lists of problems for the students to engage.

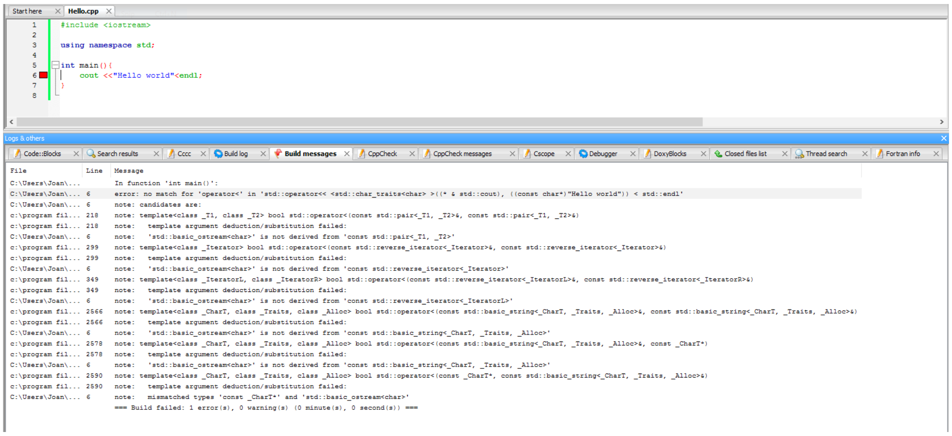
It is worth pointing out that, in general, all of these languages and platforms are student focused. There are few tools for the teachers, and this makes using only one language more difficult. There are some positive exceptions, like the case of *Jutge.org***.** (Jutge 2008–2016, Giménez *et al.*, 2012) This online automated judge has been prepared as a tool for the classroom and not just for self-learning. Teachers can set up classes, have their own sets of problems and can view students' progress. This kind of platform contributes to the gamification of learning computer science, by giving achievements, keeping track of the number of problems solved, etc. When students perceive it as a game, they become more motivated and less frustrated.

4. Our Proposal: eSeeCode

After the analysis of the pros and cons of the different languages we decided that we should try to create a language that would eliminate almost all the weaknesses.

The result of our work is eSeeCode both a language and a programming environment. Main characteristics:

- Both visual code and formal code. You can transition from one to another.



```cpp
#include <iostream>

using namespace std;

int main(){
    cout <<"Hello world"<endl;
}
```

```
File                Line   Message
C:\Users\Joan\...          In function 'int main()':
C:\Users\Joan\... 6        error: no match for 'operator<' in 'std::operator<< <std::char_traits<char> >((* & std::cout), ((const char*)"Hello world")) < std::endl'
C:\Users\Joan\... 6        note: candidates are:
c:\program fil... 218      note: template<class _T1, class _T2> bool std::operator<(const std::pair<_T1, _T2>&, const std::pair<_T1, _T2>&)
c:\program fil... 218      note:   template argument deduction/substitution failed:
C:\Users\Joan\... 6        note:   'std::basic_ostream<char>' is not derived from 'const std::pair<_T1, _T2>'
c:\program fil... 299      note: template<class _Iterator> bool std::operator<(const std::reverse_iterator<_Iterator>&, const std::reverse_iterator<_Iterator>&)
c:\program fil... 299      note:   template argument deduction/substitution failed:
C:\Users\Joan\... 6        note:   'std::basic_ostream<char>' is not derived from 'const std::reverse_iterator<_Iterator>'
c:\program fil... 349      note: template<class _IteratorL, class _IteratorR> bool std::operator<(const std::reverse_iterator<_IteratorL>&, const std::reverse_iterator<_IteratorR>&)
c:\program fil... 349      note:   template argument deduction/substitution failed:
C:\Users\Joan\... 6        note:   'std::basic_ostream<char>' is not derived from 'const std::reverse_iterator<_IteratorL>'
c:\program fil... 2566     note: template<class _CharT, class _Traits, class _Alloc> bool std::operator<(const std::basic_string<_CharT, _Traits, _Alloc>&, const std::basic_string<_CharT, _Traits, _Alloc>&)
c:\program fil... 2566     note:   template argument deduction/substitution failed:
C:\Users\Joan\... 6        note:   'std::basic_ostream<char>' is not derived from 'const std::basic_string<_CharT, _Traits, _Alloc>'
c:\program fil... 2578     note: template<class _CharT, class _Traits, class _Alloc> bool std::operator<(const std::basic_string<_CharT, _Traits, _Alloc>&, const _CharT*)
c:\program fil... 2578     note:   template argument deduction/substitution failed:
C:\Users\Joan\... 6        note:   'std::basic_ostream<char>' is not derived from 'const std::basic_string<_CharT, _Traits, _Alloc>'
c:\program fil... 2590     note: template<class _CharT, class _Traits, class _Alloc> bool std::operator<(const _CharT*, const std::basic_string<_CharT, _Traits, _Alloc>&)
c:\program fil... 2590     note:   template argument deduction/substitution failed:
C:\Users\Joan\... 6        note:   mismatched types 'const _CharT*' and 'std::basic_ostream<char>'
                           === Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

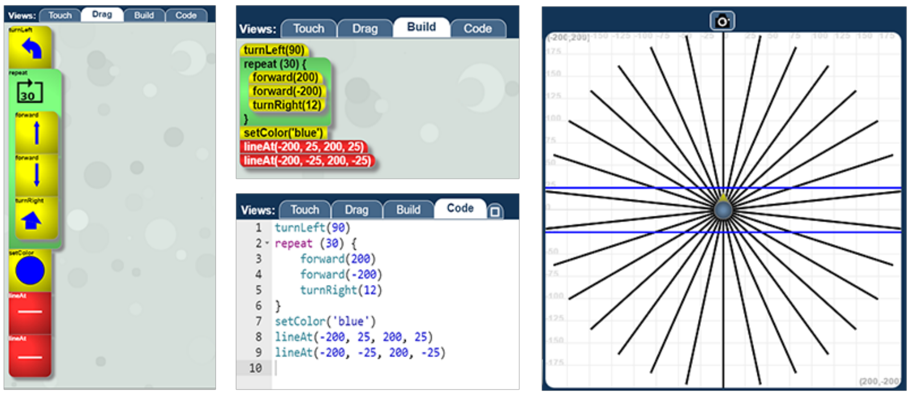
Fig. 5. Syntax error in C++. The programming environment is CodeBlocks.

- Both educational and professional language.
- It is problem solving oriented.
- Main academic areas cover basic and advanced algorithms, 2D geometry (turtle graphics) and numerical problems.
- Has a debugger, and includes simple syntax errors handling.

4.1. *Description of eSeeCode*

eSeeCode's environment offers four different programming levels: from a pure graphical click-and-run interface to a pure text syntax highlighting editor, with two middle interfaces. We call this levels *views*, because we want to show the students that code is just a representation that can have different forms. This can be seen in Fig. 6. This allows for a smooth progress in programming learning while keeping a common general interface, instruction set and platform. Time saved in this manner can be spent reinforcing other important objectives or learning a complementary second language.

The Touch view is our approach to the Puzzle Solving problems and is designed to work with students of ages 5–8. The set of instructions is represented by a set of icons (Fig. 7). The icons at this level have no text to maximize ease of use. These instructions



```
turnLeft(90)
repeat (30) {
    forward(200)
    forward(-200)
    turnRight(12)
}
setColor('blue')
lineAt(-200, 25, 200, 25)
lineAt(-200, -25, 200, -25)
```

Fig. 6. Different views of the same code and its result.

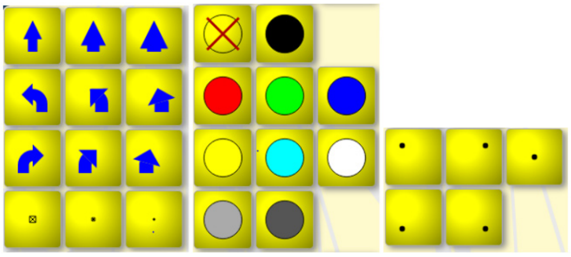


Fig. 7. Instruction set of the Touch view.

cover the basic movement of the guide, size of the drawing, colours and general positioning. Each time the student clicks one instruction the environment executes it directly.

The second view created is the Drag view. This view still uses icons as instructions but the student can move them freely once placed in the coding area. These instructions accept different arguments and the icons change to match the values of the parameters. Some examples of this behaviour can be seen in Fig. 8. Notice that the icons also include a name to help the student "read" the code. To execute a program the student needs to click the run button.

The next view is the Build view, which is similar to the Drag view as the blocks can be displaced around the code area freely. However these blocks don't include icons, just the name of the instructions and the arguments. The student can read the code fully, but to create it has to choose among a specific set available. This set is larger than the one from the drag view and contains a greater variety of instructions. This list of instructions allows the student to be able to explore eSeeCode by him/herselves, and provides the student with the familiarization of the names without having to memorize the commands and the arguments.

The last view is the Code view. In this level students can type their own code. We created eSeeCode based on JavaScript (although this base is well hidden), so after mastering in the programming in Code view the students can program freely using this well-known programming language. The platform accepts and executes any JavaScript program allowing for a deeper learning. A side advantage to the use of JavaScript is the fact that it is not required to be installed to function, as it will work with any browser.

In the Code view syntax errors are possible, but we try to give short errors that the student can correct. This can be seen in Fig. 9. The platform also has a debugger to be used in case the student's program does not execute as expected. When running a program, the editor will restyle the code to encourage students to use clean code.

In the context of the "low-floor, high-ceiling" proposed by Papert (Papert, 1980) and used by Resnick (Resnick *et al.*, 2009), eSeeCode has a lower-floor (easy-to-use) than Logo and Scratch, and a much higher ceiling (being able to hold complex programs) comparable to that of C++. The Touch view could be considered our low-floor while the Code view our high-ceiling.

Although you can try to create long programs, we have designed eSeeCode to be a problem-solving tool and have provided it with an optional easy-to-use Input/Output interface.

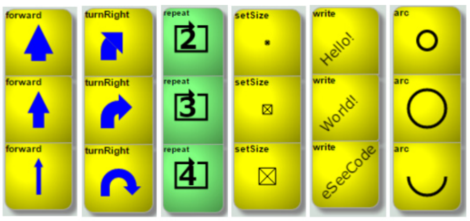
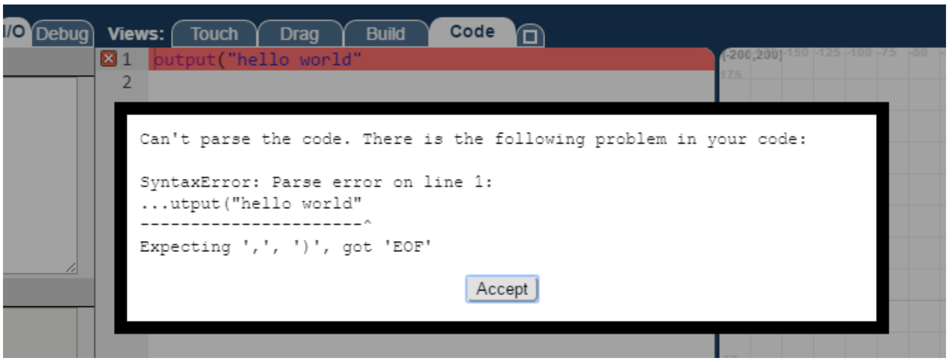


Fig. 8. The icons of the Drag view adapt depending on their parameters.



```
output("hello world"
```

```
Can't parse the code. There is the following problem in your code:

SyntaxError: Parse error on line 1:
...utput("hello world"
----------------------^
Expecting ',', ')', got 'EOF'
```

Fig. 9. Syntax error in eSeeCode.

4.2. *Teacher Point of View*

Programming languages, platforms, and materials are only oriented towards the student's use, obviating that one of the most influential parts on the learning of any process is the teacher. Providing teacher tools, and configurable platforms is key to an excellent use in class, given the uniqueness of each environment.

One of the main purposes of creating eSeeCode was also to be able to give the teacher a set of tools so s/he can create high quality exercises and materials for their courses. The interface is highly and easily configurable and it can be embedded in any webpage, allowing the teachers to configure it for each problem if needed. The tools already implemented are:

- A tutorial creation assistant, which creates dynamic tutorials
- A problem setter assistant, with which the teacher can restrict the views that can be used, decide which instructions are allowed (and how many times each can be used), preload code (hidden or not to the student) so that the student only needs to complete part of it, etc.
- Create step-by-step animations of the execution of programs.

Some of this tools are complemented by a Moodle module that allows the teacher to collect students problems and to set up specific exercises.

4.3. *Experiences with Students*

Many experiences have been carried out with students, both in an academic context and as an extracurricular activity.

eSeeCode has been used as a language to transition from Scratch to C++, and avoid the difficulties that appear from going from a visual block based language to a textual based language (Dorling *et al.,* 2015). This experience was done with 12 years old students that had previous knowledge of Computer Science since they had taken some

Scratch courses. This allowed the teachers to teach directly using Code view, but they allowed the students to go back to the Build view to avoid the empty page difficulty (Resnick *et al.*, 2009), where the student doesn't know how to start because he is used to having a set of blocks. With the introduction of eSeeCode the teachers had to review the basic concepts of variables, conditionals, loops, etc. The methodology that was used was based on Polya problem-solving principles, where the student should take the following steps (Polya, 1945):

1. Understanding the problem *i.e. Can you explain the problem with different words? Can you create your own "paper and pencil" examples?*
2. Devising a plan. *In our case this is clearly the Algorithm. Some things to consider when devising a plan are to try to find previous, similar problems.*
3. Carrying out the plan. *This is the implementation of the algorithm. We propose the "baby steps" methodology, where you write the code step-by-step and execute along to avoid syntax errors, while making sure everything goes accordingly to the plan.*
4. Looking Back. *Although no judge system has been created, the student should analyse if s/he obtained the desired result, going back to previous steps if s/he did not.*

Although this experience is different than the experiment done by Lewis in a study to compare Scratch vs Logo (Lewis, 2010), a similar survey was created and it was taken by 59 of the students in the course. The survey consisted of 16 questions each being a 4-level Likert scale.

As it can be seen in Fig. 10 it seems that Scratch is easier to program, but in fact the total number of students that have a positive feedback (Strongly agree and Agree)

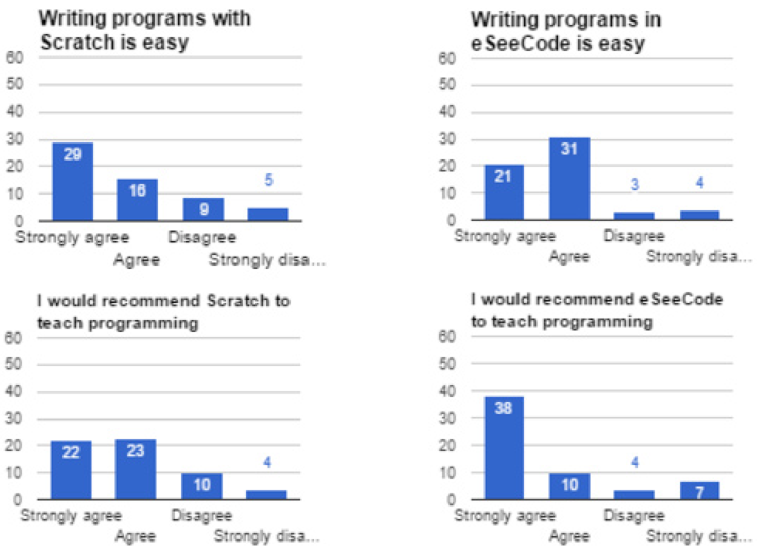


Fig. 10. Students responses to a 4-level likert test about the use of Scratch and eSeeCode in class.

is larger in the case of eSeeCode. One of the difficulties to analyse this question is the fact that the problems that the students had to solve in the two languages are different. What is interesting is that students would recommend in general to use eSeeCode to teach programming, this might have similar reasons to what Lewis describes (Lewis, 2010), as the students feel more self-secure when typing their own code, and might see Scratch as something different than programming. This would be interesting to analyze in a future study.

When asking questions about the difficulties when learning the language, we encounter different opinions depending on the topic. In Fig. 11 we can see this results. Similar to what happened when asking about *writing a program* the opinion of the students is less strong with eSeeCode than with Scratch, although the numbers of positive vs negative are similar. We have to take into account that the view students were using more is the Code view, which makes it difficult to give a precise analysis of the situation.

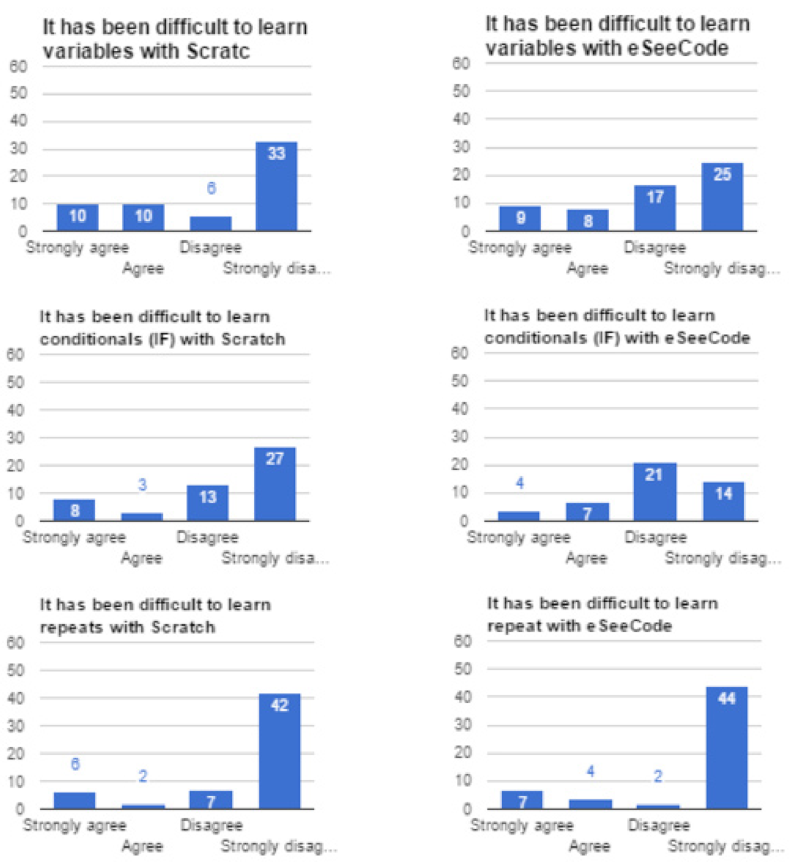


Fig. 11. Student responses to a 4-level likert test about the difficulties when learning with Scratch and eSeeCode.

What might be interesting is that the students opinion about the difficulties of the concept of *repeat* is statistically independent of the language (a Fisher exact test gives us a p-value of 0.3401 which tells us that the groups are not significantly different) This can be because the *repeat* concept is very easy to understand by the students.

Another experience we want to share is our own version of hour of code (Fig. 12) where students had to make programs to draw some typical optical illusions. Placing this activity in the context of students having an enjoyable time seemed to motivate the students to complete the tutorials and to try to draw their own images. Some of the activities were designed to be more difficult than the level of knowledge the students possessed and were accompanied with a solution code. The students would read the code and try to figure out the expected result. Although some students were not able to complete the activity, most of them enjoyed it. Another experiment we conducted was to give students the partial code for the program. In this activity we eliminated all the numbers from the code. The student was required to fill the gaps, until the right image would appear. Very few students would try numbers at random, the majority would first try to understand what the code did, and place the right numbers directly.

Three different sets of students tried the platform with this activity: Students that had never programmed before, students that had been introduced to Scratch and students that already knew Logo and C++. The difficulties found were similar in each group, concluding that it adapts to the student's needs.

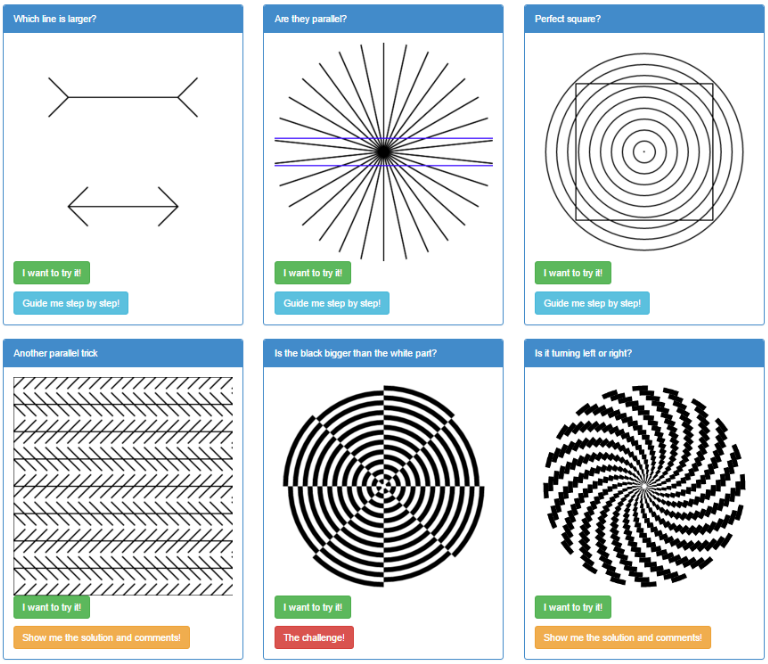


Fig. 12. Tutorials from our Hour of Code.

4.4. *Further Work*

We have released eSeeCode as an open source project, and as such it is a continuing development. We believe that further work in the platform should include:

- A new and adaptable design to make it feel more modern, and more user-friendly.
- Not allowing syntax errors in the Blocks and Build views.
- Teacher support material to be able to be used in class by non-programmer teachers.
- A formal study of the impact in the long run process and how it can be included in the regular curriculum. We believe this study should contain reports on Scratch, C++ and eSeeCode.

5. Conclusions

The time has come for teachers in Spain to take on the responsibility of curriculum development. This responsibility will come first by understanding the different options there exist, understanding the previous objectives, the ones we want to have in their place, and taking on a global vision. Right now one language cannot satisfy all the learning process. It is also valuable for the students to know more than one language, which would provide the option of overcoming the inherent weaknesses of each one.

eSeeCode tries to provide a new platform to overcome the main weaknesses found, but we believe it does not need to be a replacement but rather a complement to the learning process. The trials so far show that it is a viable language to take into the classroom, and that the students show a good overall satisfaction with it.

6. References

Ackovska, N., Erdösné NéMeth, Á., Stankov, E., Jovanov, M. (2015). Report of the IOI Workshop "Creating an International Informatics Curriculum for Primary and High School Education". *Olympiads in Informatics*, 9, 205–212.

Boytchev, P. (2014). *Logo Tree Project.* `http://www.elica.net/download/papers/LogoTreeProject.pdf`

Code.org (2013). `http://www.code.org`

Dijkstra, E. (1999). Computing Science: achievements and challenges. *ACM SIGAPP Applied Computing*, 7(2), 2–9.

Dorling, M. (2014). Computer progression pathways. *Computing at Schools.* `http://www.computingatschool.org.uk`

Dorling, M., White, D. (2015). Scratch: a way to Logo and Python. In: *SIGCSE '15: Proceedings of the 46th ACM Technical Symposium on Computer Science Education.*

Duncan, C., Bell, T. (2015). A pilot computer science and programming course for primary schools. In: *Proceeding WiPSCE '15 Proceedings of the Workshop in Primary and Secondary Computing Education.* 39–48.

Duke, R., Salzman, E., Burmeister, J., Poon, J., Murray, L. (2000). Teaching programming to beginners – choosing the language is just the first step. In: *ACSE '00: Proceedings of the Australasian conference on Computing education.*

eSeeCode (2015–2016). `http://www.eseecode.com`

Finkel, D., Hooker C., Salvidio, S., Sullivan, M., Thomas C. (1994). Teaching C++ to high school students. In:

SIGCSE '94: Proceedings of the twenty-fifth SIGCSE symposium on Computer science education.

FMSLogo (2006–2016). `http://fmslogo.sourceforge.net/`

Giménez, O., Petit, J., Roura, S. (2012). Jutge.org: an educational programming judge. In: *Proc. of the 43rd ACM Technical Symposium on Computer Science Education* (SIGCSE-2012). 445–450.

Gouws, L.A., Bradshaw, K., Wentworth, P. (2013). Computational thinking in educational activities: an evaluation of the educational game Light-bot. In: *ITiCSE '13: Proceedings of the 18th ACM conference on Innovation and technology in computer science education.*

Harvey, B., Mönig, J., (2010) Bringing "No Ceiling" to Scratch: can one language serve kids and computer scientists? *Constructionism 2010, Paris.* `http://www.eecs.berkeley.edu/~bh/BYOB.pdf`

Jutge.org (2012). `http://www.jutge.org`

Lewis C. (2010). How programming environment shapes perception, learning and goals: logo vs. Scratch. In: *SIGCSE '10: Proceedings of the 41st ACM technical symposium on Computer science education.* 346–350.

Light-Bot (2016). `http://www.lightbot.com`

Papert, S. (1980). *Mindstorms: Children, Computers and Powerful Ideas.* Basic Books. Inc. USA.

Polya, G. (1945). *How to solve it.* Princeton University Press, USA.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, A., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM*, 52(11).

Robins, A., Rountree, N. (2003). Learning and teaching programming: a review and discussio*n. Journal Computer Science Educations*, 13(2), 137–172.

Rubinstein, R. (1974). Using Logo in teaching. *ACM SIGCUE Outlook*, 9(SI).

Saez-Lopez, J.M., Roman-Gonzalez, M., Vazquez-Cano, E., (2016). Visual programming languages integrated across the curriculum in elementary school: a two year case study using "Scratch" in five schools. *Elsevier computers & Education,* 97, 129–141.

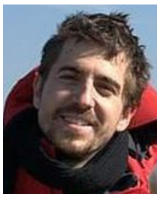
Scratch (2013–2016). `https://scratch.mit.edu`

Simon, J. (1996). *L'evolució de l'ensenyament del llenguatge logo a l'escola de mestres Blanquerna 10 anys.* VI Seminari Logo Terrassa, Spain.

Stroustrup, B. (2007). Evolving a language in and for the real world: C++ 1991–2006. In: *HOPL III: Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages.* ACM.

Winslow, L. (1996). Programming pedagogy – a psychological overview. *ACM SIGCSE Bulletin*, 28(3).

J. Alemany Flos holds a degree in mathematics and is currently completing a Master's Degree in Advanced Mathematics and Mathematical Engineering. He is a former high school teacher, who now designs and helps develop programming curricula in schools as a consultant. He has been involved with the National Informatics Olympiad for the last eight years, also attending the International Olympiad in Informatics as a team leader and deputy leader since 2010.



J. Vilella Vilahur holds a degree in Computer Science and has been teaching programming and robotics for the last ten years at Aula Escola Europea where he also provides the IT support. He has been involved in several open source projects and has been the lead programmer in eSeeCode.