Evolving technology to better support teaching introductory programming inside *Moodle*

Leônidas de Oliveira Brandão*, Igor Moreira Félix*, Patricia Alves Pereira‡ and Anarosa Alves Franco Brandão§
*Instituto de Matemática e Estatística - IME
University of São Paulo
São Paulo, Brazil
Email: leo@ime.usp.br, igormf@ime.usp.br
‡ Ibmec São Paulo
São Paulo, Brazil
Email: patricia.pereira@ibmec.edu.br
§Escola Politécnica
University of São Paulo
São Paulo, Brazil
Email: anarosa.brandao@poli.usp.br

***Abstract*—Nowadays, it is increasing the importance of developing skills related to the computational thinking at earlier stages of education, and the adoption of tools that implement the visual programming paradigm has been well succeeded in presenting introductory notions of programming from kindergarten to college. Such tools allows the user to program while manipulating blocks that represent programming languages instructions. *iVProg* is among this class of tools and was firstly implemented in Java and could be integrated to *Moodle* using *iAssign*. Due to technological evolution, *iVProg* also evolved to *iVProgH* and its integration to *Moodle* is still possible by using the new version of *iAssign* that allow the integration of *HTML 5* packages to *Moodle*. In this paper we describe how this evolution takes place and presents *iVProgH* functionalities as well as *iAssign* extension to include *HTML 5* packages into *Moodle*. In addition, we give some teaching support whenever integrated using *iVProgH* with *Moodle* using *iAssign*.**

***Abstract*—A importância de se desenvolver as habilidades relacionadas ao pensamento computacional tem ganho importância nos últimos anos, tendo recebido crescente atenção mesmo nos estágios iniciais da educação formal. Além disso, a adoção de ferramentas que implementam o paradigma de programação visual tem apresentado relativo sucesso para se introduzir o noções iniciais de programação, principalmente no ensino fundamental I. Tais ferramentas permitem que o usuário-aprendiz programe a partir da manipulação de ícones ou blocos que representam as intruções das linguagens de programação. Entre essas ferramentas, encontra-se o *iVProg*, cuja primeira versão foi implementada em Java em 2009, possibilitando seu uso como aplicativo ou integrado à ambientes *Web*. Desde a primeira versão o *iVProg* também podia ser integrado ao ambiente *Moodle* a partir do uso do pacote *iAssign*. Nesse artigo apresentamos uma necessária evolução do *iVProg* para uma versão em *HTML 5*, o *iVProgH*. Para manter a integração com o *Moodle*, foi preciso também avançar o pacote *iAssign*, que agora permite a integração de qualquer pacote de módulo interativo de aprendizagem codificado em *HTML 5*.**

Index Terms*—*iVProg*, *iAssign*, visual programming, *Moodle

I. Introduction

The increasing role of Information and Communication Technologies in our daily lives as well as the pervasiveness of computing in several branches of activities makes computational thinking a strong desirable skill. Such a need brings the inclusion of introductory programming disciplines in different education levels. In fact, the discipline of introductory programming is part of other curriculum than the STEM one, but also of humanities and medicine. In addition, it was also included in elementary and high school [1, 2].

Computational thinking involves the ability of representing and solving a problem in a way that could be "easily" translated in a programming language [3, 4]. This is why introducing the first notions of programming at earlier stages of education is so important.

Nevertheless, for those who were at college or universities at the first decade of the 2000, that are numbers to attest the fail rate in introductory programming courses was about 33% [5, 6, 7]. Bosse and Gerosa [8] report that more than 50% of students enrolled in MAC115, an introductory course of programming at USP, during the years of 2010 to 2014, had failed.

Although the numbers are recent, this is not a novelty: Du Boulay [9] stated in 1986 that "Learning to program is not easy". Moreover, there is an awareness that the lack of skills and cognitive abilities to formulate and solve problems algorithmically within novices result in high fail and drop out rates.

In this context, the proposition of approaches to facilitate the acquisition of such skills and cognitive abilities is strongly desirable. It is worth to mention that acquiring such skills and abilities is beyond learning an specific programming language like C, Java or Phyton. The visual programming paradigm is one of the approaches adopted to teach and learn algorithms, which are computational solutions for problems or, in other words, description of computational thinking. We can cite, among existing visual programming tools, Alice [10], Greenfoot [11], Scratch [12] and *iVProg* [13].

By adopting visual programming to teach introductory programming, we use graphical components to allow the construction of algorithms while simply manipulating them and reducing the cognitive load related to the syntax of traditional programming languages [10, 11, 14, 15, 16]. Moreover, due to its nature of stimulating computational thinking, tools to support visual programming has been successfully used even in advanced programming courses, such as distributed programming [16].

Whenever we think about introducing algorithms for novices at universities and colleges, it is of particular interest that tools to support the teaching process could be available anywhere, anytime, integrated to web-based learning environments, such as *Moodle*.

In this paper we present *iVProgH*, the *HTML 5*[1] version of *iVProg*, and an extension of *iAssign* [17] to incorporate *interactive Learning Modules (iLM)* [18] coded in *HTML 5* into *Moodle*. These are technological evolution of existing tools to address new constraints of web browsers.

The paper is structured as follows: section II briefly presents *iVProg* and *iAssign* in their earlier versions, section III presents *iVProgH* and its model to perform automatic assessment, section IV presents the extension of *iAssign* to allow the integration of *interactive Learning Modules (iLM)* coded in *HTML 5* in *Moodle*. In addition, section V presents some guidelines to use *iVProgH* integrated to *Moodle* and, finally, in section VI we present our final considerations and conclusion.

II. *iVProg* AND *iAssign*

In this section we present two of the *LInE*[2] free educational systems, the *iVProg* and the *iAssign*. The first one is an educational software to be used to present the concepts of programming, and the second is a software to integrate other educational *Web* software to *Moodle*, when these software could be classified as *interactive Learning Modules (iLM)* [18, 19, 20, 21]. Besides, all *iVProg* versions can be considered *iLM*, so they can be integrated to *Moodle* by the *iAssign*.

A. *iVProg*

iVProg[3] (interactive Visual Programming in the internet) is a tool to support teaching and learning of introductory programming. Its first version was launched in 2009 [14] and a refactored version is available since 2015 [22]. The conception of *iVProg* was guided by the following requirements:

- allow its integration to a web-based system or platform to support teaching and learning;
- allow the authoring of exercises to be deployed in repositories;
- provide automatic evaluation of exercises;
- decrease the cognitive load of learning the syntax of an specific programming language while learning algorithms.

[1]This means *HTML* (*Hypertext Markup Language*) version 5, enriched by *JavaScript* and *CSS* (*Cascading Style Sheets*).

[2]Laboratory of Informatics in Education, sited on the University of São Paulo - `http://line.ime.usp.br`.

[3]*iVProg*: available at http://www.matematica.br/ivprog/

The first version of *iVProg* was a simplification of Alice 2.0 [10]. By simplification we mean extracting some functionalities, such as 3D animation, to allow its use as an *applet*[4]. A screenshot of its interface is presented in figure 1, with an algorithm to calculate $n!$ (the factorial of a non negative integer n). This version could be executed within a web browser (*applet*) or individually, as a Java application.

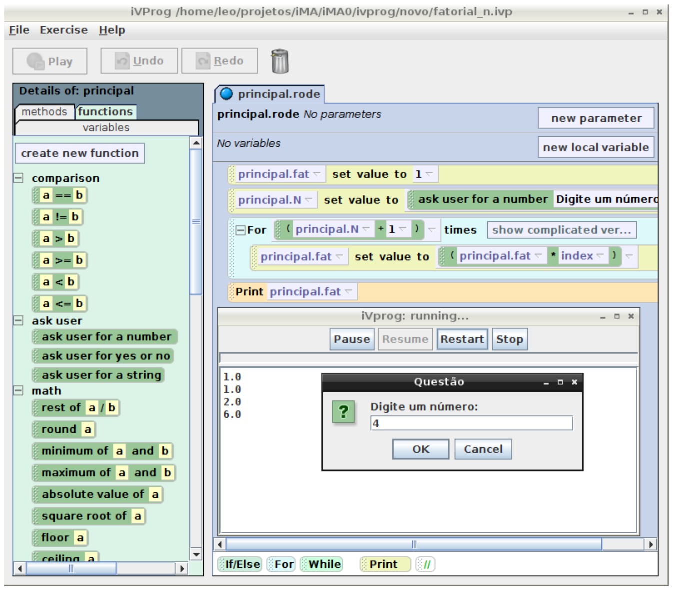


Figure 1. *iVProg* (Java) - version 1

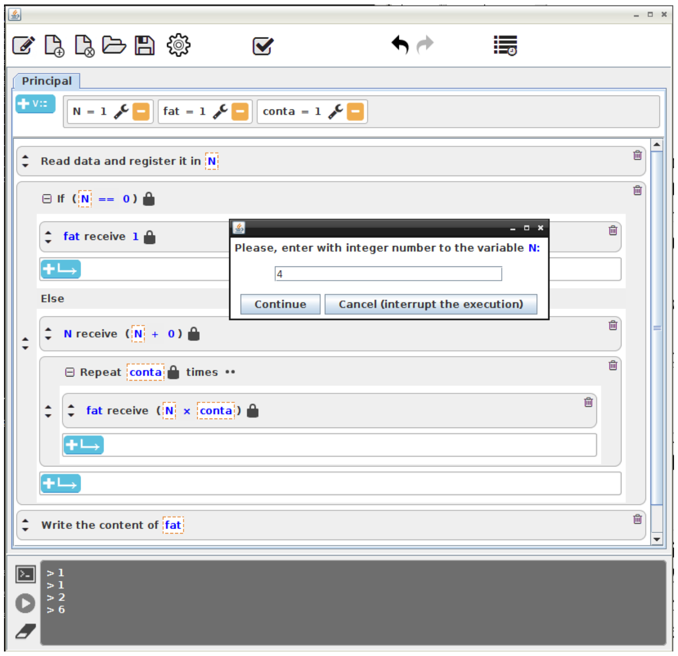


Figure 2. *iVProg* (Java) - version 2.

The second version of *iVProg* was a refactored version of the first one. In fact, it was conceived as a product of an *iLM* software product line [21]. The interface of this

[4]mini *Java* applications that had used to run in any web browser.

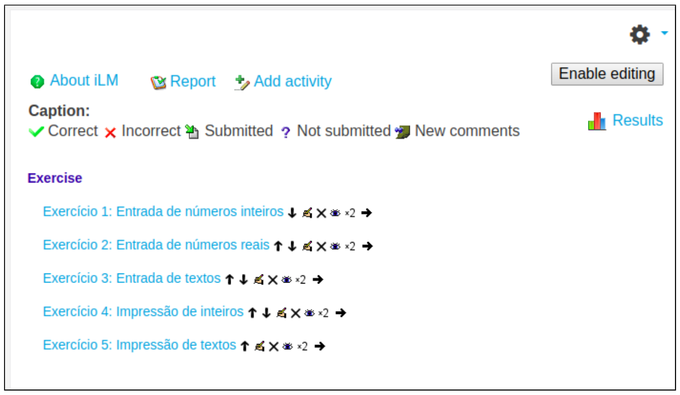
version is presented in figure 2, with the same example presented in figure 1. Its interface includes the *commands* in a contextualized way, differently from its first version and *Alice*.

Since it was first launched, *iVProg* has been successfully used by teachers and students in introductory programming courses [13], most of the time integrated with *Moodle* using *iAssign*. However, since 2015 some web browsers started experiencing problems to run *applets*. This was because it depends on the *NPAPI (Netscape Plug-in API)* technology to properly run and, for instance, *Firefox/Mozilla* does not include NPAPI since version 52, *Chrome* since version 45.

For this reason we started a new *iVProg* version, now implemented in *HTML 5*, and initiated the extension of *iAssign* to allow the incorporation of *iLM* coded in *HTML 5* to *Moodle*. The *iVProg* in *HTML 5* is presented in section III and the extension of *iAssign* is presented in section V.

B. *iAssign*

iAssign is one of the avaliable modules to extends the *Moodle* system. It brought to *Moodle* the main idea introduced by *SAW (Web Learning System)* [18, 19, 20] package with resources for improving its interactivity by allowing the integration of *iLM* into educational contexts offered by the system. The main functionalities provided by *iAssign* are: interactive activity, *iLM* filter, and detailed report [17]. A screenshot of the *iAssign* interface is given in figure 3.



Caption:
✓ Correct ✗ Incorrect Submitted ? Not submitted New comments

Exercise

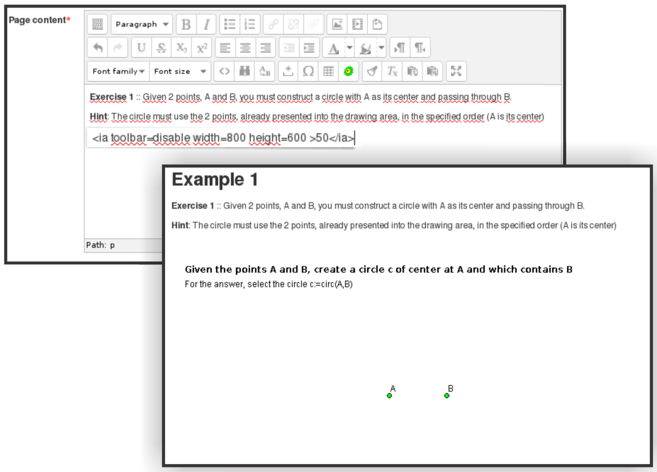
Exercício 1: Entrada de números inteiros ×2
Exercício 2: Entrada de números reais ×2
Exercício 3: Entrada de textos ×2
Exercício 4: Impressão de inteiros ×2
Exercício 5: Impressão de textos ×2

Figure 3. *iAssign* activities' main interface

An *Interactive Activity* is a resource for authoring interactive activities (exercises or examples) using *iLM*; an *iLM* filter is a resource for integrating *iLM* to *Moodle*, making possible to use the *iLM* in any *Moodle*'s asynchronous context, such as forum, glossary, wiki etc.

Figure 4 shows the use of an *iLM* to support teaching Geometry in the context of a *glossary*. At the top of figure 4 we have the authoring interface of the *Moodle*'s glossary activity. In order to include the *iLM* to turn the activity into an interactive activity it is needed to insert the *iLM* file name between specific tags delimiters (<ia> $\cdots$ </ia>) where it must be displayed within the text. The bottom of figure 4 shows the way the interactive activity is displayed for students.

A detailed report provides information related to the performance of students while doing the interactive activities.



Exercise 1 :: Given 2 points, A and B, you must construct a circle with A as its center and passing through B.

Hint: The circle must use the 2 points, already presented into the drawing area, in the specified order (A is its center)

```
<ia toolbar=disable width=800 height=600 >50</ia>
```

Example 1

Exercise 1 :: Given 2 points, A and B, you must construct a circle with A as its center and passing through B.

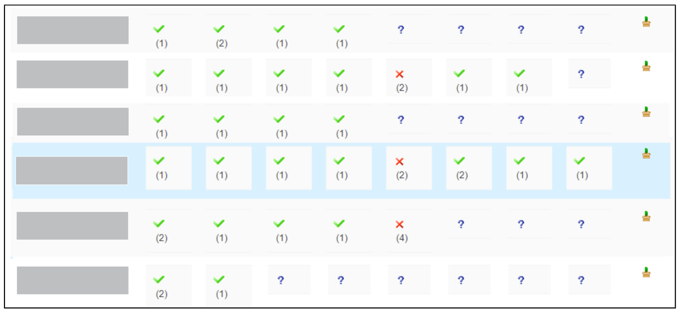
Hint: The circle must use the 2 points, already presented into the drawing area, in the specified order (A is its center)

Given the points A and B, create a circle c of center at A and which contains B

For the answer, select the circle c:=circ(A,B)

Figure 4. *iLM* filter - including an Interactive Activity of Geometry.

It includes information about how many attempts the student got to finish the activity and, if the *iLM* provides automatic assessment resources, it presents the status of the activity, e.g, if it was evaluated as correct or wrong (see figure 5).



	✓ (1)	✓ (2)	✓ (1)	✓ (1)	?	?	?	?
	✓ (1)	✓ (1)	✓ (1)	✓ (1)	✗ (2)	✓ (1)	✓ (1)	?
	✓ (1)	✓ (1)	✓ (1)	✓ (1)	?	?	?	?
	✓ (1)	✓ (1)	✓ (1)	✓ (1)	✗ (2)	✓ (2)	✓ (1)	✓ (1)
	✓ (2)	✓ (1)	✓ (1)	✓ (1)	✗ (4)	?	?	?
	✓ (2)	✓ (1)	?	?	?	?	?	?

Figure 5. Detailed report of activities (green tick means correct; red wrong)

III. *iVProgH*

iVProgH is an evolving implementation of *iVProg* version 2, now in *HTML 5*, to provide interoperability among all current *Web* browsers. Figure 6 presents its main interface, which is very similar to the previous version (see figure 2). In addition, *iVProgH* can be easily incorporated to the any web-based environment by simply writing a few lines of HTML code.

iVProgH maintains the *commands* contextualized, as it was in *iVProg* version 2. This model brings as advantage to learners the reduction of options available in the graphical interface. For example, in an initial context, any command is available (see left block in figure 7), but after creating a *selection* command, the options available in the context of *selection* are automatically restricted to the viable ones, which mean, the creation of *logical expressions* (see the right side in figure 7).

In the *iVProgH* stable version, the following types of variables are available: integer, real, text and *boolean*. In addition, it provides resources to attributions, two variations of loops (*while* and *for*), commands for control flow (logical

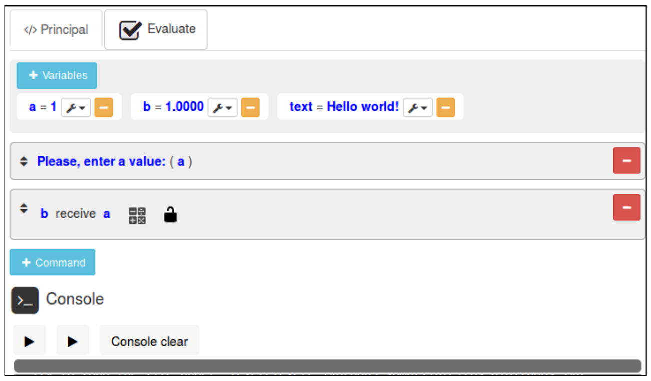


Figure 6. *iVProgH* main interface

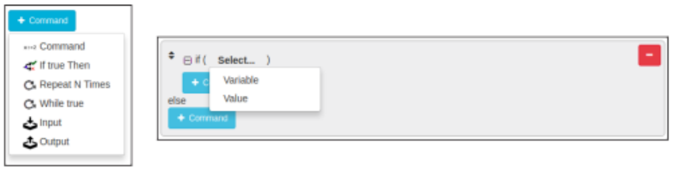


Figure 7. Contextualized commands in *iVProgH*

selection) and command for input/output (i.e., read and print). Comparing both versions, *iVProgH* innovates in the authoring interface provided to teachers, to create interactive activities with automatic assessment.

Automated assessment in *iVProgH* is implemented based on test cases. Therefore, in order to prepare an exercise to have automatic assessment, the teacher must create a list of inputs and their corresponding outputs, the *expected outputs*. The *expected outputs* are used to compare the values provided by the one who solve the exercise and press the bottom to send it for evaluation.

1) Test Case design: The aim of test cases is to simulate the behavior of a correct algorithm. For example, in an activity targeting the output of the average of two integer values (entered by the user), one test case could be: using as inputs the list $(-1, 102)$ and as its corresponding output the single value 51.5 (i.e. $(-1+102)/2 = 101/2$). A second test could have as input the values -1 and 101, and the correspondent output 50.

In formal terms, each test case consists of two lists, the *input list* $(e_1, e_2, \ldots, e_k)$ and its correspondent (if using a correct algorithm) *output list* $(s_{1,1}, s_{1,2}, \ldots, s_{1,l_1},\ s_{2,1}, s_{2,2}, \ldots, s_{i,l_2},$ $\ldots\ s_{k,1}, s_{k,2}, \ldots, s_{k,l_k})$. On one hand, when the student's algorithm is tested, *iVProgH* uses e_1 as its first input, e_2 as the second input, and so on. On the other hand, if $(o_{1,1}, o_{1,2}, \ldots, o_{1,l_1},$ $o_{2,1}, o_{2,2}, \ldots, o_{i,l_2},\ \ldots\ o_{k,1}, o_{k,2}, \ldots, o_{k,l_k})$ are the outputs from the student's algorithm when using the *input list*, these are compared with the *expected output* (some distance metric like $\sum_{i=1}^{k}\sum_{j=1}^{l_i}\|o_{i,j} - s_{i,j}\|$).

To avoid mistakes, it is recommended that the teacher implements a complete version of the expected algorithm to create the list of inputs and *expected results* to the activity. Having the algorithm, the teacher must provides the most significant inputs to the problem and submits them to the it. The *output list* will be formed by the algorithm's outputs, in the very same order. The process of generating a list of test cases is represented in figure 8.

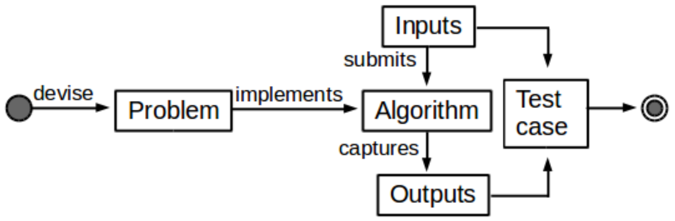


Figure 8. Schema to Generate a Test Case

It's possible to illustrate the use of test cases using a simple problem: *Develop an algorithm that reads an integer value and prints this number and its square*. To this problem, the test cases could be the following six pair of lists (table I):

Table I
EXAMPLE OF TEST CASES

Inputs	Outputs	
-5	-5	25
-2	-2	4
0	0	0
8	8	64
10	10	100
100	100	10000

2) Solving an exercise in iVProgH: creating and algorithm: The students must have access to the problem statement, devise a solution for it, implement an algorithm in *iVProg* (that solves the problem) and then send it to evaluation. This process is represented in figure 9.

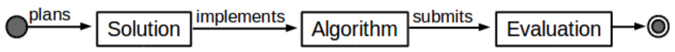


Figure 9. Algorithm Elaboration and Submission Process

When the student finishes one version of algorithm-solution and "press" evaluation button, the *iVProg* proceeds to the following steps: 1. gets the first item e_1, from the first test case, and submits it to student's algorithm; 2. captures the outputs generated by the input $(o_{1,1}, o_{1,2}, \ldots, o_{1,l_1})$; and 3. compares them to the expected outputs $(s_{1,1}, s_{1,2}, \ldots, s_{1,l_1})$.

Depends on the *iLM* configuration, it is possible to evaluate if the algorithm is correct without sending it to the server, as many times as the student wish. In current version of *iAssign*, only the last student's submission (and its grade) is registered.

3) Automatic Assessment in iVProgH: As aforementioned, *iVProgH* compares the outputs sent after the solution of an exercise with the expected ones, resulting in a grade between 0 and 1. If there is no difference between the expected output and the sent output, it is attributed integral grade 1. However if significant difference is detected in the sent outputs, the grade should be zero (0). *iVProgH* repeats this process for all entries and, at the end of the list of test cases, it generates a mean of grades.

The assessment process is presented in diagram at the figure 10. The algorithm is executed using the input list $(e_1, \ldots, e_k)$. Then the sent outputs $(o_{1_1}, \ldots, o_{k,l_k})$ are compared with the expected outputs $(s_{1_1}, \ldots, s_{k,l_k})$. The automatic assessment process begins with the reception of the algorithm sent for evaluation, then it is executed for the whole list of test cases inputs. Finally, the comparison among the generated outputs and the *expected results* and its associate results are presented ("printed") in the *iVProgH* output area.

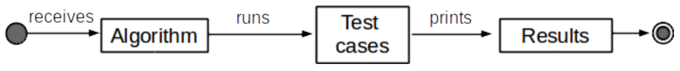


Figure 10. Algorithm Evaluation Process

It's important to highlight that the test case evaluation's efficiency is directly related to their coverage extension. This mean that teachers must provide test cases for each possible variations for the algorithm behavior. For instance, if the purpose of an algorithm is to determine the highest value in a sequence, it is necessary that at least one test case includes the highest number in the first position and other test must have the highest number at the end of the sequence.

IV. Integrator of *iLM* to *Moodle* via *iAssign*

Since the first version of *iAssign*, in 2010, it provides the integration of *Java* packages to the *Moodle* environment. This means that any *Moodle* administrator can install new *iLM*, teachers can produce new interactive activities to their students with this *iLM*, and students can view, test and send their solutions, in an integrated fashion. Besides, *iAssign* has usual features to import, export, duplicate, remove, and configure. In the figure 11 is presented a components diagram for *Moodle-iAssign-iLM*. In the diagram we can devise that *iAssign* is integrated with *Moodle* by its *Activity modules* and *Filters*.

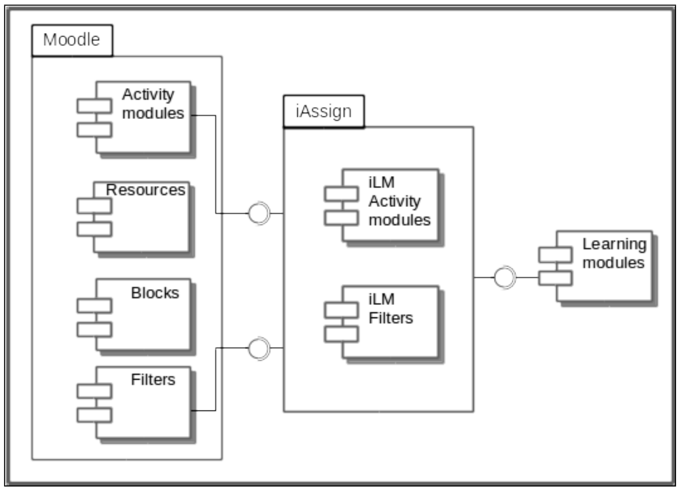


Figure 11. Components of *Moodle-iAssign-iLM*

To allow the incorporation of *HTML 5*, new *PHP* scripts were developed and others were re-factored, resulting in 32 new or modified scripts, from a total of 51. The current directory structure is presented in table II.

Table II
The *iAssign* directory structure

Directories	Comments
mod/iassign	*iAssign* directory inside *Moodle*
mod/iassign/backup/moodle2	containing files to provide backups/restoration
mod/iassign/classes/event	treatment to event
mod/iassign/db	database management (updates, upgrades, …)
mod/iassign/icon	support imagens
mod/iassign/ilm	directory to all the installed *iLM*
mod/iassign/ilm_debug	to help debug during development
mod/iassign/ilm_handlers	new directory to manage *iLM Java* and *HTML 5*
mod/iassign/lang	language dictionary (PT_BR, EN_US, FR)
mod/iassign/pix	to provide *iAssign* logo

To be integrated into *Moodle*, an *HTML 5* package, as well as any other *iLM*, has to implement at least the first two methods described below:

- a method to read a content file through *URL* (so *iAssign* can inform the parameter `iLM_PARAM_Assignment` to the *iLM* package);
- a method named `getAnswer()`, to returns to *iAssign* a text with the student's answer to the activity; and
- if the *iLM* contains an engine to automatic assessment, a method named `getEvaluation()`.

If the *iLM* implements automatic assessment, like *iVProgH*, *iAssign* manages the results obtained by student in the activity, inserting the activity's grade in the *Moodle gradebook*.

For security reasons, the integration of new *iLM* is restricted to users with administrative permissions in *Moodle* system. To add a new instance or a new version of a previously installed *iLM*, the user accesses the administrative area, enables block *edition*, selects the tab *plugins* and, in the block *activities*, selects the package *iAssign*. Figure 12 shows the *Moodle* interface for such installation.

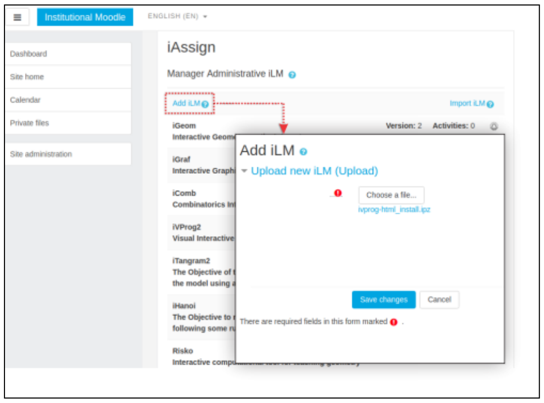


Figure 12. Adding a new *iLM* in *Moodle* via *iAssign*

When the new *iLM* has been successfully installed, users with *teacher*'s role in some course, are able to create new activities using the recent added *iLM* and their students may solve such activities.

A. *iLM integrated to Moodle: operation*

When integrated into *Moodle*, any available *iLM* is ready to be used. The teacher can author interactive activities using the *iLM* and make them available to the students. In their turn, the students can solve proposed problems using the *iLM*, test their solution (if the *iLM* provides automatic assessment mechanism), and sent it to the server. The teacher can see the student's answer. If the *iLM* has a mechanism to automatic assessment, then *iAssign* also manages the associated grades, integrating them to the *Moodle*'s *gradebook* (or *Grader report*).

It is worth note that, *iAssign* considers as grade a value between 0 and 1 (0.5 means medium grade) that is registered in the *Moodle* database.

V. *iVProgH* INTEGRATED TO *Moodle* VIA *iAssign*

Any *iLM* can be considered as a coarse-grained *learning object (LO)* since they can be found, reused, and interoperable [23, 24]. Besides, the essence of LO is to be devoted to education and be used via *Web*. Several *Learning Management System (LMS)*, like *Moodle*, provides methods to allow the LO integration and usage.

In this sense, the *iAssign* is a *Moodle* integrator of *iLM*, like *iVProgH*. However its use is tight integrated, as explained in previous sections. Moreover, since *iVProgH* provides automatic assessment, the teacher can easily detect conceptual flaws or, even, if an activity is not well adjusted (maybe presenting some conception error or the difficulty is over the students knowledge). For instance, in a period of time, if no student could solved the problem, this could mean that the statement is ambiguous or contains errors.

To the students, a significant advantage of *iVProgH-iAssign* over a weak integrated LO could be the facility to use and detect problems in their solution. In the following, we will describe how an activity can be authored using *iVProgH-iAssign*, as well as how this activity can be executed. A sequence diagram of this operation is given in Figure 13.

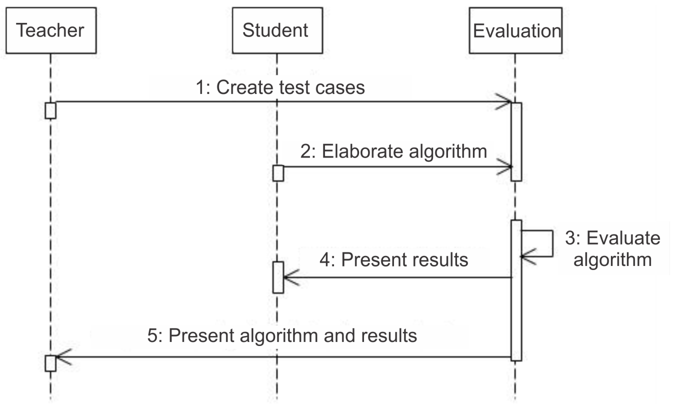
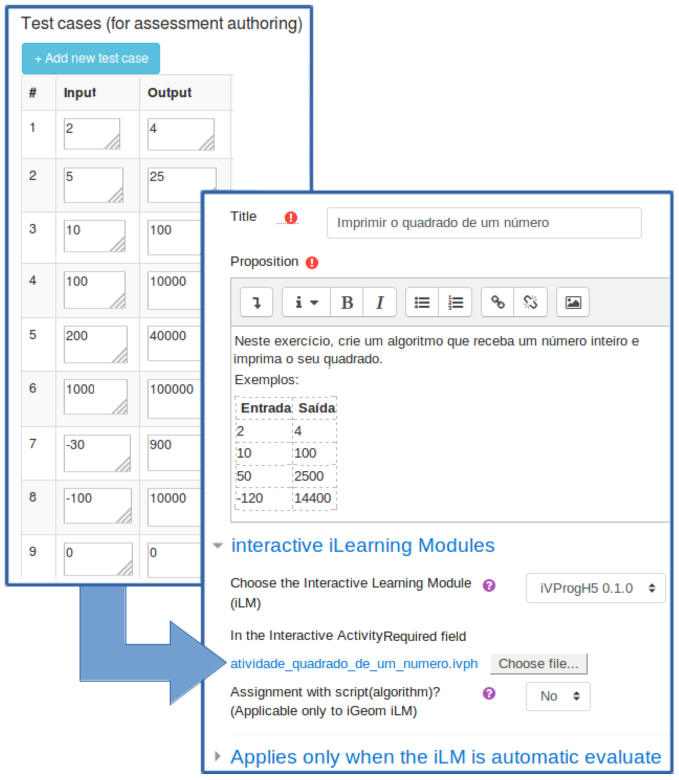


Figure 13. *iVProg* Operation Sequence Diagram

1) Activity authoring - teacher role: The *iVProgH* provides an interface for teachers to create activities, actually to create test cases, as explained in section III. The initial step is the common one to any *Moodle* activity creation (turning on the *edition* mode; selecting the *Add an activity or resource* and *iAssign* option). After the *iAssign* block of activity is created (as in figure 3), it must be selected inside it the option *Add activity* and chosen the *iVProgH* among the *iLM* options. In this context, it must be opened the *online editor*, that will result in the test case builder in figure 14, the left rectangle.



Test cases (for assessment authoring)

+ Add new test case

#	Input	Output
1	2	4
2	5	25
3	10	100
4	100	10000
5	200	40000
6	1000	100000
7	-30	900
8	-100	10000
9	0	0

Title: Imprimir o quadrado de um número

Proposition

Neste exercício, crie um algoritmo que receba um número inteiro e imprima o seu quadrado.
Exemplos:

Entrada	Saída
2	4
10	100
50	2500
-120	14400

interactive iLearning Modules

Choose the Interactive Learning Module (iLM): iVProgH5 0.1.0

In the Interactive ActivityRequired field: atividade_quadrado_de_um_numero.ivph Choose file...

Assignment with script(algorithm)? (Applicable only to iGeom iLM): No

Applies only when the iLM is automatic evaluate

Figure 14. Activity authoring

2) Working out on an activity - student role: The activity is available to the student in a *Web* page with the *iLM*, and the *iLM* loads the corresponding activity file. In the *iVProg* example, the file with its test cases.

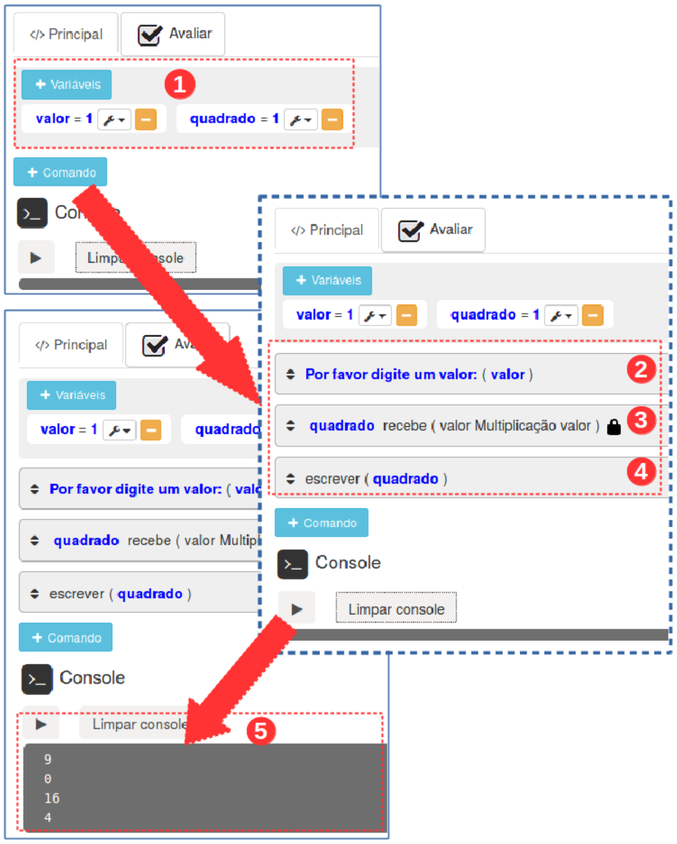
Using as example the problem: *Build an algorithm that, after reading an integer number, it prints the square of this number*, an example of solution is presented in figure 15.

If the student wants, it is possible to run the algorithm, by "clicking" the *run button* (represented as a black triangle in the figure 15). The output is presented at the console area (the gray area, at the botton of the figure).

VI. FINAL CONSIDERATIONS

In this paper we presented the evolution of a suite of tools to support the teaching and learning algorithms and introductory programming in web-based learning environments such as *Moodle*.

The suite of tools is composed of an *iLM* and a *Moodle* package for integrating *iLMs* to *Moodle*. The *iLM* is a visual



```
9
0
16
4
```

Figure 15. Creating and testing an algorithm

programming tool with automatic assessment resources, the *iVProgH*, and the integrator is *iAssign*. This suite evolution aims to provide interoperability among all existing web browsers and avoid problems with incompatibility concerning legacy Java applets. Therefore, *iVProgH* is implemented in *HTML 5* and *iAssign* was extended to integrate *HTML 5* modules to *Moodle*.

This evolution willl be extended to all the available *iLM* in order to support interactive learning of issues like Geometry, Calculus, and Combinatorics.

References

[1] F. Heintz, L. Mannila, K. Nygårds, P. Parnes, , and B. Regnell, "Computing at school in sweden - experiences from introducing computer science within existing subjects," in *Proceedings of the International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*. Cham: Springer, 2015. [Online]. Available: *URL*https://lucris.lub.lu.se/ws/files/1573415/8033906.pdf

[2] V. Dagiene, T. Jevsikova, C. Schulte, S. Sentance, and N. Thota, "A comparison of current trends within computer science teaching in school in germany and the uk," in *Proceedings of the 6th International Conference ISSEP*, ser. ISSEP 2013, 2013, pp. 63–75.

[3] J. M. Wing, "Computational thinking," *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, 2006.

[4] V. Barr and C. Stephenson, "Bringing computational thinking to k-12: what is involved and what is the role of the computer science education community?" *ACM Inroads*, vol. 2, no. 1, pp. 48–54, 2011.

[5] M. E. Caspersen, "Educating novices in the skills of programming," Ph.D. dissertation, PhD thesis, Department of Computer Science, Universidade de Aarhus, Dinamarca, 2007.

[6] J. Bennedsen and M. E. Caspersen, "Failure rates in introductory programming," *SIGCSE Bulletin*, vol. 39, no. 2, pp. 32–36, Jun. 2007. [Online]. Available: *URL*http://doi.acm.org/10.1145/1272848.1272879

[7] C. Watson and F. W. Li, "Failure rates in introductory programming revisited," in *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, ser. ITiCSE '14. New York, NY, USA: ACM, 2014, pp. 39–44. [Online]. Available: *URL*http://doi.acm.org/10.1145/2591708.2591749

[8] Y. Bosse and M. A. Gerosa, "Reprovações e Trancamentos nas Disciplinas de Introdução à Programação da Universidade de São Paulo: Um Estudo Preliminar," in *WEI-Workshop sobre Educação em Computação.(2015)*, 2015, pp. 1–10.

[9] B. D. Boulay, "Some difficulties of learning to program," *Journal of Educational Computing Research*, vol. 2, no. 1, pp. 57–73, 1986. [Online]. Available: *URL*https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9

[10] S. Cooper, W. Dann, and R. Pausch, "Alice: A 3-D Tool for Introductory Programming Concepts," in *Proceedings of the Fifth Annual CCSC Northeastern Conference on The Journal of Computing in Small Colleges*, ser. CCSC '00. USA: Consortium for Computing Sciences in Colleges, 2000, pp. 107–116. [Online]. Available: *URL*http://dl.acm.org/citation.cfm?id=364132.364161

[11] P. Henriksen and M. Kölling, "Greenfoot: Combining object visualisation with interaction," in *Companion to the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications*, ser. OOPSLA '04. New York, NY, USA: ACM, 2004, pp. 73–82. [Online]. Available: *URL*http://doi.acm.org/10.1145/1028664.1028701

[12] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: Programming for All," *Commun. ACM*, vol. 52, no. 11, pp. 60–67, Nov. 2009. [Online]. Available: *URL*http://doi.acm.org/10.1145/1592761.1592779

[13] L. de Oliveira Brandão, R. da Silva Ribeiro, and A. A. Brandão, "A system to help teaching and learning algorithms," in *Frontiers in Education Conference (FIE), 2012*. IEEE, 2012, pp. 1–6.

[14] R. R. Kamiya and L. de Oliveira Brandão, "ivprog - um sistema para introdução à programação através de um modelo visual na internet," *Anais do XX Simpósio Brasileiro de Informática na Educação. Florianópolis, SC*, 2009.

[15] L. de Oliveira Brandão, Brandão, A. A. Franco, and R. da Silva Ribeiro, “ivprog – uma ferramenta de programaçao visual para o ensino de algoritmos,” in *Anais dos Workshops do Congresso Brasileiro de Informática na Educação*, vol. 1, 2012.

[16] B. Broll, A. Lédeczi, P. Volgyesi, J. Sallai, M. Maroti, A. Carrillo, S. L. Weeden-Wright, C. Vanags, J. D. Swartz, and M. Lu, “A visual programming environment for learning distributed programming,” in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE ’17. New York, NY, USA: ACM, 2017, pp. 81–86. [Online]. Available: *URL*http://doi.acm.org/10.1145/3017680.3017741

[17] P. A. Rodrigues, L. de Oliveira Brandão, and A. A. F. B. ao, “Interactive assignment: A Moodle component to enrich the learning process,” in *2010 IEEE Frontiers in Education Conference (FIE)*, Oct 2010, pp. T4F–1–T4F–6.

[18] L. de Oliveira Brandão, S. Isotani, and J. G. Moura, “A plug-in based adaptive system: Saaw,” in *International Conference on Intelligent Tutoring Systems*, August 2004, pp. 791–793.

[19] S. Isotani and L. de Oliveira Brandão, “Ferramenta de avaliação automática no igeom,” in *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE*, vol. 1, no. 1, November 2004, pp. 319–328.

[20] J. G. Moura, L. de Oliveira Brandão, and A. A. F. B. ao, “A web-based learning management system with automatic assessment resources,” in *2007 IEEE Frontiers in Education Conference (FIE)*, October 2007, pp. F2D–1.

[21] D. L. Dalmon and L. de Oliveira Brandão, “Uma linha de produto de software para módulos de aprendizagem interativa,” *Simpósio Brasileiro de Informática na Educação (SBIE)*, 2012.

[22] R. da Silva Ribeiro, “Construção e uso de ambiente visual para o ensino de programação introdutória,” Ph.D. dissertation, Master dissertation, Universidade de São Paulo, 2015.

[23] R. McGreal, “Learning objects: a practical definition,” *International Journal of Instructional Technology & Distance Learning*, vol. 1, no. 9, 2004. [Online]. Available: *URL*http://www.itdl.org/journal/sep_04/index.htm

[24] J. Braga, *Objetos de aprendizagem: Introdução e fundamentos*. UFABC, 2015, vol. 1.