Towards a Framework to Scaffold Problem-solving Skills in Learning Computer Programming

Gorgoumack Sambe[1], Khadim Drame[1] and Adrien Basse[2]
[1]*University Assane SECK, Ziguinchor, Senegal*
[2]*University Alioune DIOP, Bambey, Senegal*

Keywords: Programming Learning, Problem Solving Skills, Explicit Guidance, Semantic Analysis.

Abstract: Developing problem solving skills through learning programming has become a real challenge. Problem-solving skills are fundamental to learning computer programming and can be developed during learning. Teachers focus more on the syntax of the languages than on the development of problem solving skills. We present a conceptual framework to promote problem-solving skills in learning computer programming. This framework is based on an IDE which integrates two components. The first one is an explicit guidance to support the acquisition of skills related to different stages of a problem-solving method. It consists in explicitly following the steps of the process with activities that develop related skills. The second one is a semantic feedback system to develop problem-solving skills.

1 INTRODUCTION

Learning computer programming has become a necessity not only for future computer scientists but for everyone. Indeed, such learning helps to develop problem solving skills and system design competence and thus makes it possible to confront real life problem (Einhorn, 2012).

However, research shows that at the end of the introduction to programming, many learners face difficulties when it comes to problem solving. Weakness of problem-solving skills and their non-integration into the learning process is one of the main factors of drop-out (Luxton-Reilly et al., 2018; Medeiros et al., 2018).

Developing problem solving skills during the learning process is a challenge. Teachers tend to focus on the syntax of the language rather than on problem-solving skills (de Raadt, 2007; Lister et al., 2004; McCracken et al., 2001). Yet, Problem-solving skills are essential in learning computer programming. Thus, they should be promoted and developed during the learning process (De Raadt et al., 2009; Muller & Haberman, 2009; Sprankle & Hubbard, 2008).

In face-to-face teaching of computer programming, many strategies have been used, which can have a positive impact on problem solving skills. One of the strategies consists in explicitly guiding learners through the steps of the problem-solving method. The other one provides learners with feedback on semantics of the code (Muller & Haberman, 2009; Sprankle & Hubbard, 2008).

In the context of Computer Based Learning Environment (CBLE), related work addresses a different issue even though the problem solving aspect is less taken into account.

Based on this finding and on the strategies used in face-to-face teaching, we propose a platform to promote problem-solving skills in programming learning. This platform is based, on the one hand, on explicitly guiding learners through the steps of the problem-solving process; and on the other hand, on feedback on semantics of the code

Section 2 of the paper provides the background of our framework: problem-solving skills. In section 3, we present related work to scaffold problem-solving skills in a learning environment Section 4 presents the scaffolding framework. Section 5 concludes the paper with suggestions for future research.

2 PROBLEM SOLVING IN LEARNING PROGRAMMING

Problem-solving is defined by (D'Zurilla & Nezu, 1988) as the "conscious effort in controlled

Sambe, G., Drame, K. and Basse, A.
Towards a Framework to Scaffold Problem-solving Skills in Learning Computer Programming.
DOI: 10.5220/0010446503230330
In *Proceedings of the 13th International Conference on Computer Supported Education (CSEDU 2021) - Volume 1*, pages 323-330
ISBN: 978-989-758-502-9
Copyright © 2021 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

information processing that is aimed at identifying, discovering, or inventing a solution to a problem". In computer programming, problem solving is often seen as an iterative process made up of steps that are made up of a set of skills. In (McCracken et al., 2001), the authors adopt an iterative five-step problem-solving framework:

1. Problem Comprehension/Abstract the problem from its description;
2. Generate sub-problems;
3. Transform sub-problems into sub-solutions;
4. Recompose the sub-solutions into a working program;
5. Evaluate and iterate.

Problem solving skills are essential for learning programming and lack of problem solving skills has been identified as the major cause of learners' failure in introduction to programming (Nelson et al., 2017). Such skills can be developed during learning. Each of these steps requires a set of skills and the development of problem solving skills consists in developing the skills for each step.

Among the strategies to develop these skills, there are the explicit guidance and the feedback on code semantics (Muller & Haberman, 2009; Oh et al., 2017). The explicit guidance consists in explicitly following the steps of the process with activities that develop the skills related to it. For example, the activity of reformulation of the problem help to develop skills related to abstraction and understanding of the problem. The following table shows some skills and their associated supported activities for a course designed to help learners to develop problem solving skills in a face-to-face teaching (Belhaoues et al., 2016; Muller & Haberman, 2009).

Table 1: Some problem solving skills and their associated activities.

Problem-solving Skill	Learning/Instructional Activity
Problem's comprehension	Reformulation of the problem statement in terms of initial state, goal, assumptions and constraints
Problem's decomposition	Identifying, naming and listing subtasks.
Analogical reasoning	Identifying similarities between problems. Distinction between structural and surface similarities. Raising awareness of common mistakes caused by referencing to unsuitable problems.
Generalization and abstraction	Extracting prototypes of problems from analogical problems in different contexts.
Identifying problem's prototype	Problems' categorization
Problem's structure identification	Identifying the relation between subtasks

3 RELATED WORK

In the context of CBLE, most of related work address issues such as ontologies, assessment and sometimes problem solving. However, explicit guidance in the problem solving process and feedback on semantics of the code are addressed.

The explicit guidance in problem solving approach consists in explicitly following the steps of the process with activities that develop related skills. In (Sambe & Basse, 2020), the author propose a method that introduces the learner to a three-step problem-solving process: the first step allows to identify the input data and their corresponding types; the second step allows to identify the output data and their types; in the last step, the learner have to sequence the previously mixed source code of an expert solution to the problem. This work is an extension of the work of (Diatta et al., 2019, 2018) which propose ontologies, based on algoskills (Belhaoues et al., 2016). The proposed ontologies represent algorithmic exercises and their solutions in pseudo-code. It also allow to automatically generate instances of a program in one of the proposed languages from an instance of pseudo code.

About **semantic feedback**, we can distinguish:

1. **Dynamic approaches** where programs are executed and assessed using a battery of unit tests. Programs are run with standard input to check if they produce the expected output;
2. **Static approaches** where programs are not executed but analysed by looking at the source code instructions. Among the **static approaches**, there are
   - **Manual code analysis;**
   - **Automatic code analysis.**

In **dynamic approaches**, the EPFL (Ecole Polytechnique Fédérale de Lausanne) grader is an automated grader to assess students' assignments during the MOOCs course "Introduction to programming with C++" (Bey et al., 2018). Submissions of learners are compiled and unit-tested

over a set of inputs. Learners receive an automatic feedback on how their code performed in the tests.

In **static approaches with manual code** analysis, Algo+ is an automated assessment tool of programs using program matching (Bey & Bensebaa, 2011). Submission of a learner is assessed by comparing it to a set of predefined solutions already assessed by an instructor. Predefined solutions are those which are detected as common and frequent in learners' submissions. They contain correct programs but also erroneous ones. Instructors give a feedback to predefined solutions (correct and erroneous) that are stored and used as a source of learning and evaluation for new cases submitted. The feedback given by the instructor should be general, semantic or not.

In **static approaches with automatic analysis of code**, (Broisin & Hérouard, 2019) introduce an indicator that estimates the value of the edit distance between two scripts. First, they turn a program into an abstract syntax tree (AST) and then into a string of tokens (command, assignation, loop,…). They adapt the edit distance of levenshtein between two strings of characters to the strings of tokens. They observe a correlation between the value of the indicator and the score assigned by a human tutor: the higher the human score of a program formulating a solution to a problem, the smaller the distance between that program and the solution of the problem.

We note that most related work is oriented in a specific field such as ontologies or automatic program assessment, while barely integrating problem-solving. Regarding semantic comparison and feedback, as Broisin says, "works on this line are still scarce and very few solutions have been proposed".

4 PROPOSITION

Our long-term goal is to set up a platform that supports the learner in the development of problem-solving skills. The platform will be an integrated development environment (IDE) much more oriented towards pedagogy unlike traditional IDE. Our system seeks to develop problem-solving skills by helping the learner to follow the steps of the problem-solving process on the one hand and on the other hand by providing feedback on the semantics of the code after the edition of functional code.

In a technical aspect, the platform will be set up in web access for online exercises allowing us to keep traces of learners. It will also be available as a standalone environment for offline exercise. We have a database of problems proposed by experts in the field with expert solutions. Problems given include sequential, conditional and iterative problems for different levels of difficulty. The expert solution includes the number of output data and their type, the number of input data and their type and the functional code proposed by the expert. For all problems in the database, our system calculates and stores the semantic value of the expert solution to the problem.

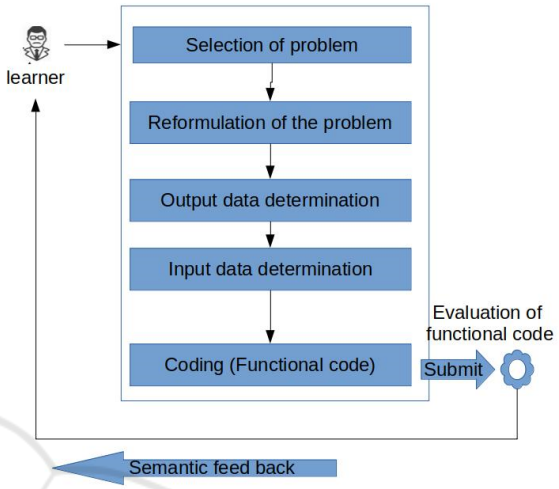

Figure 1: Process of problem solving on our system.

4.1 Explicit Guidance in Problem Solving Approach

Our platform does not allow, at the beginning of training, to directly edit source code, the learner has to follow a four-step problem-solving process. This is how, after selecting a problem to solve, the learner has to follow these different steps before being able to edit code:

1. **Reformulation of the Problem:** we remind that the exercise of reformulating a problem with feedback is an activity that develops skills related to understanding the problem;
2. **Identification of the Output Data and Their Type:** in this activity, learners have to propose the number of output data and the type of each output data;
3. **Identification of the Input Data and Their Type:** in this activity , learners have to propose the number of input data and the type of each input data;
4. **Edition of the Functional Code:** Once the first three steps are validated, learners can edit their source code. From the expert's solution, our tool offers the source code for entering inputs and displaying outputs, the learner will only have to edit the functional code.

The reformulation of the problem, the entry of the number of input data and the number of output data by the learner are done using forms with input fields

Let's consider the problem of exchanging the value of two integer variables that we call P1. Our platform asks the learner to reformulate the problem and then to give the number and type of output variables and the number and type of input variables. The learner is expected to reformulate the problem and to provide two integer input data and two integer output data. Since our application framework is the Pascal language, our system generates the code below and lets the learner edit the functional code.

```
Program exchange_two_integer;
   {input/output data}
   var x,y:integer;
   {intermediate variables}
   t : integer;
begin
   {reading of input data}
   write('give the value of x : ');
   readln(x);
   write('give the value of y: ');
   readln(y);
   {put your functional code here}

   {writing of output data}
   writeln('x=', x,', y=', y);
   readln();
End.
```

Let's take another example with the problem of calculating the sum and the average of three integer variables that we call P2. The learner is expected to reformulate the problem and to provide three integer input data, one integer output data (sum) and one real output data (mean). The following code is generated by our system and the learner has to edit the functional code.

```
Program mean_three_integer;
   {output data}
   Var s: integer;m:real;
   {input data}
  i1,i2,i3 :integer;
begin
   {reading of input data}
   write('give the first number : ');
   readln(i1);
   write('give the second number :');
   readln(i2);
   write('give the third number : ');
   readln(i3);
   {put your functional code here}

   {writing of output data}
   writeln('the sum is : ', s);
   writeln('the mean is : ', m);
   readln();
End.
```

4.2 Semantic Comparison of Source Codes

Regarding the feedback on the semantics of the functional source code, we propose a system that makes a semantic comparison of the learner's code with the expert source code. We introduce here the semantic comparison system. In our system, we are interested in input variables with known semantics, solving the problem by finding the output values with the correct semantics.

Our system assigns literal values to program input variables, and through a process of chaining instructions and algebra, it determines final semantic values of the output variables. We remind that in imperative programming, introductory courses are concerned with assignment, sequence, condition and iteration. In this first version, we were interested in the assignment and the sequence of instructions.

4.2.1 Semantic Value

For each problem P, we define Ip, the set of input data and Op the set of output data proposed by an expert. We define concepts below:

- **Initial Semantic Value of an Input Data:** it is a literal set for input data by our system in the same type of the variable. For example, for the first problem, the three input data are set to their semantic values which are literals:
  SEM(i1)=l1;
  SEM(i2)=l2;
  SEM(i3)=l3.
  For the second problem of exchanging the value of two integer variables, the semantic value of this two integer variables are set to SEM(x)=l SEM (y)=m.
- **Semantic Value of a Variable:** it is the value of the variable expressed in functions of the input literals, at a step t of execution of the functional code of the program Pr. For a variable v in a program Pr, we denoted it $SEM_{Pr}(v)$. For example, for the variable s affected by the assignation s:=i1+i2 in a program Pr, the semantic value of s is SEM_{Pr} (s)=SEM(i1+i2)=l1+l2
- **Semantic Value of a Program:** the semantic value of a program Pr is the semantic value of the set of the output data of the problem at the end of the program. For example, for the problem of calculating sum and average of three integer variables, the semantic value of a program Pr1 proposed for this problem is the semantic value of sum and average at the end of the program. For the problem of exchanging two variables x and y,

the semantic value of the program Pr2 is the semantic value of x and y at the end of the program.
SEM(Pr1)={SEM_{Pr1}(sum),SEM_{Pr1}(mean)}
SEM (Pr2)={SEM_{Pr2}(x),SEM_{Pr2}(y)}

- **Semantic Equality/Inequality of Programs:** two programs Prog and Prog', proposed for a problem P, are equal when they have the same semantic value i.e. the same semantic value for the output data. Therefore, two programs are unequal when their semantic value are unequal.

4.2.2 Processing of the Semantic Value of a Variable/Program

We remind that we are concerned here with the assignment and the sequence of instructions, especially a sequence of assignment. At the beginning of a program, all input variables are set with a literal called semantic value of the input data. The functional code is a sequence of assignments that runs sequentially. An assignment is an instruction that sets a value of the impacted variable by the value of the expression. An expression is a finite combination of symbols organized according to rules that depend on the context. During initiation, symbols can designate constants, variables, arithmetic, algebraic or logical operations and groupings to determine the order of priority of operations (parentheses). Operators are unary or binary.

Semantic Value of an Expression. The semantic value of an expression is obtained by replacing each symbols participating to the expression by its semantic value.

Proof:

let us call **SEM(expr)** the semantic value of the expression **expr**.

Expressions are composed by symbols which are constants or variables and operations.

For any input data **x** which is a variable such that the semantic value is set to a literal **l**, the semantic value of the expression **x** is **SEM(x)=l**. (it's not necessary here to break it down for each simple type introduced in initiation : integer, real, boolean and char).

For any unary operator **unaryop** and an input data x such that the semantic value is set to a literal l, the semantic value of the expression unaryop x is equal to unaryop l therefore

SEM(unaryop x) = unaryop SEM(x). For any binary operator **binaryop** and input data x and y such that the semantic value is set respectively to literals l and m, the semantic value of the expression x binaryop y is equal to l binaryop m therefore

SEM(x binaryop y)= SEM(x) binaryop SEM(y)

For any constant C, the semantic value of the constant is **SEM(C)=C**. In the same way as for an input data, we demonstrate that

SEM(unaryop C)= unaryop SEM(C) = unaryop C and that

SEM(C binaryop y)= SEM(C) binaryop SEM(y) =C binaryop m

Based on the fact that an expression is based on variables that are already set, we can generalize that for any expression. The semantic value of the expression is equal to the expression where all variables and constants are replaced with their semantic value.

End of the Proof. From this proposition, we can obtain the semantic value of a program/variable by chaining instructions. To do so, we just have to replace the variables participating in the expressions by their semantic value**.**

For example, for a program with three integer input variables i1, i2 and i3 and the sequence of instructions

s:=i1+i2;
s:=s*i3;

Input data are initialized with their initial semantic value:

SEM(i1)=l1
SEM(i2=l2
SEM(i3)=l3

After the first instruction s:=i1+i2, the semantic value of s is :

SEM(s)=SEM(i1)+SEM(i2)=l1+l2

After the second instruction s:=s*i3, the semantic value of s is

SEM(s)=SEM(s)*SEM(i3)=(l1+l2)*l3

We can see that this sequence of instructions and the instruction s:=i3*i2+i3*i1 have the same semantics but with a different writing.

To get around this difficulty, we just have to show that the operators for the four types (integer, real, char, boolean) introduced in initiation keep their properties in the semantic value of the expression.

Property of Operators. Operators keep their mathematical properties in the semantic value of an expression

Proof:

Let us consider the operator of arithmetic addition. For two symbols (real or integer) x and y with the semantic value l and m,

SEM(x+y)= SEM(x)+SEM(y)=l+m and SEM(y+x)= m+l

l+m = m+l therefore

SEM(x)+SEM(y)=SEM(y)+SEM(x)

So, we have to prove the set of properties for all operators that exist in the language. However, we will

limit ourselves, in this document, to proving it for this single operator and will generalize it for other priorities and operators.

End of the Proof. Based on that, it's obvious that the semantic value of a variable affected by an assignation is the semantic value of the expression of the assignation. We can determine the semantic value of all variables and semantics of a program by the process of replacing each parameter of an expression by its semantic value in a sequence.

In technical aspects, we have implemented our system in python and we use multiple technologies:

- ANTLR (www.antlr.org): a powerful parser generator that we use to generate an Abstract Syntax Tree of the program.
- Sympy (www.sympy.org): a python library for symbolic mathematics that we use as a computer algebra system.

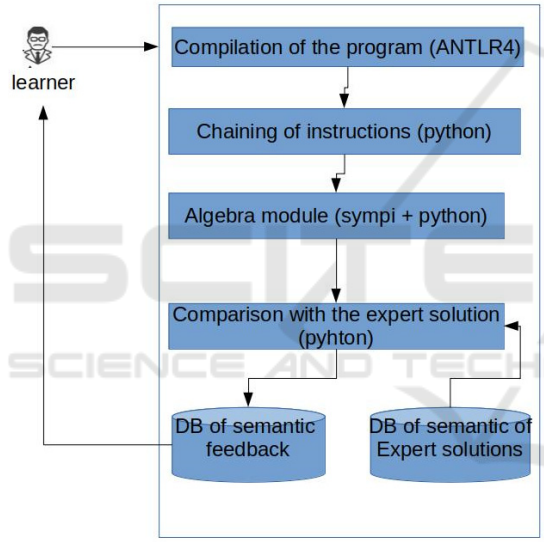

Figure 2: Evaluation process of learner's code.

When learners submit their functional codes, the evaluation system follows these steps (Figure 2: Evaluation process of learner's code):

1. First, we compile a functional code of the learner, a classical compilation, lexical and syntax verification. For this, we use the AST generated by ANTLR.
2. Then, we go through the AST to construct semantic values of variables by chaining. At this stage process, we use sympy to turn expressions into an identical form based on the properties and order of priority of the operators. This facilitates the semantic comparison of expressions and therefore variables;
3. Last, we compare the semantic value of the proposition of the learner with the semantic value of the expert solution and then give a semantic feedback to a learner.

4.3 Example of Application

4.3.1 Problems and Functional Code of the Expert

Let's take an example with two problems:

1. The problem of calculation of the sum and the arithmetic mean of three integer variables requested from the user;
2. The problem of exchanging two integer variables x and y.

The learner will first be asked by our system to reformulate the problem and then to give the number of input and output data of each of these problems and their data type:

4. For the calculation of the sum and the average, we will have two output variables: sum being integer and average being real. The three integer entries will be denoted i1, i2 and i3;
5. For the exchange of variables, we have two integer inputs x and y and two outputs which are the inputs x and y.

The source code generated by our system is given beforehand and the functional code is expected from the learner. We give here the functional code of the expert and the semantic value of the variables after each instruction.

Table 2: Functional code of the human expert.

Pr	Functional code	Processing of semantic values

E1	`s:=e1+e2+e3;`
	`m:= s/3;`
		`SEM(s)=l1+l2+l3`
		`SEM(m)=s/3=`
		`(l1+l2+l3)/3`

E2	`t:=x;`
	`x:=y;`
	`y:=t;`
		`SEM(t)=l`
		`SEM(x)=m`
		`SEM(y)=l`

The semantic of the functional code of the expert E1 for the problem P1 is
SEM(E1) ={s=l1+l2+l3, m=(l1+l2+l3)/3}
and the semantic of the program E2 is
SEM(E2)={x=m, y=l}.

4.3.2 Proposition of Learners and Semantic Value

For the first problem we have in the table some propositions of learners and the process of calculation of semantic value of the program.

Table 3: Propositions of learner for P1.

Pr	Functional code of the learner	Processing of the semantic

1	s:=i1+i2;
	s:=s+i3;
	m:=s/3
		SEM(s)= l1+l2
		SEM(s)=s+i3=(l1+l2)+l3 =l1+l2+l3
		SEM(m)=s/3=(l1+l2+l3)/3

2	i1:=i1+i2;
	s:=i1+i3;
	m:=s/3;
		SEM(i1)=l1+l2
		SEM(s)=i1+i3=l1+l2+l3
		SEM(m)=SEM(s)/3=(l1+l2+l3)/3

3	s:=i2+i3+i1; m:=s/3;	SEM(s)=l2+l3+l1 SEM(m)=s/3=(l2+l3+l1)/3
4	s:=i1+i2+i3; m:=i1+i2+i3/3	SEM(s)=l1+l2+l3 SEM(m)=l1+l2+l3/3

SEM(Pr1) ={s=l1+l2+l3, m=(l1+l2+l3)/3} and
SEM(Pr4)= {s=l1+l2+l3, m=l1+l2+l3/3}
Pr2 and Pr3 have the same semantic value as Pr1
The first three programs have the same semantic value as the expert program and the last one has a different semantic value.

Table 4: Proposition of learners for P2.

Pr	Functional code of the learner	Processing of semantic values

1	t:=x;
	x:=y;
	y:=x
		SEM(t)=l
		SEM(x)=m
		SEM(y)=l

2	x:=y; y:=x;	SEM(x)=m SEM(y)=m

3	x:=x+y;
	y:=x-y;
	x:=x-y;;
		SEM(x)=l+m
		SEM(y)=l+m-m=l
		SEM(x)=l+m-l=m

SEM(Pr1) ={x=m,y=l}

SEM(Pr2) ={x=m,y=m}

SEM(Pr3) ={x=m,y=l}

The program 1 and 3 have the same semantic value as an expert solution and program 2 which have a different semantic value is false.

Our system in this first version is limited to the evaluation of equivalence and semantic difference, we are currently working to set up the feedback base which will be based on an automation process based on an analysis of the types of semantic errors.

5 CONCLUSION

In this article, we introduce a framework for promoting problem solving skills in learning programming. This framework underpins an IDE for developing these skills by leading learners to follow a four-step problem-solving process and by giving him feedback on semantics of the code. In this first version we are interested in developing problem solving skills through learning of assignment and sequence.

This work is a first step for the implementation of an IDE, based on literature in problem solving skills in learning programming and on scaffolding of problem solving on CBLE. Our aim in the future is to implement and evaluate the impact of the IDE on problem solving skills, learning process and persistence on learning programming.

We plan an extension of this framework taking account of control and iteration. Then we will make an experiment in a real context with learners. Finally, we plan to collect data and traces and use them to validate our strategy through an analysis based on educational data mining methods.

REFERENCES

Belhaoues, T., Bensebaa, T., Abdessemed, M., & Bey, A. (2016). AlgoSkills : An ontology of Algorithmic Skills for exercises description and organization. *Journal of e-Learning and Knowledge Society*, *12*(1).

Bey, A., & Bensebaa, T. (2011). ALGO+, an assessment tool for algorithmic competencies. *2011 IEEE Global Engineering Education Conference (EDUCON)*, 941-946. https://doi.org/10.1109/EDUCON.2011.5773260

Bey, A., Jermann, P., & Dillenbourg, P. (2018). A Comparison between two automatic assessment approaches for programming : An empirical study on MOOCs. *Journal of Educational Technology & Society*, *21*(2), 259–272.

Broisin, J., & Hérouard, C. (2019). Design and evaluation of a semantic indicator for automatically supporting programming learning. *Proceedings of The 12th International Conference on Educational Data Mining (EDM 2019)*, 270–275.

de Raadt, M. (2007). A review of Australasian investigations into problem solving and the novice programmer. *Computer Science Education*, *17*(3), 201–213.

De Raadt, M., Watson, R., & Toleman, M. (2009). Teaching and assessing programming strategies explicitly. Proceedings *of the Eleventh Australasian Conference on Computing Education-Volume 95*, 45–54.

Diatta, B., Basse, A., & Ndiaye, N. M. (2018). Framework and Ontology for Modeling and Querying Algorithms. *International Conference on Interactive Collaborative Learning*, 536–544.

Diatta, B., Basse, A., & Ouya, S. (2019). PasOnto : Ontology for Learning Pascal Programming Language. *2019 IEEE Global Engineering Education Conference (EDUCON)*, 749–754.

D'Zurilla, T. J., & Nezu, A. M. (1988). On Problems, Problem Solving, Blue Devils, and Snow : A Reply to Krauskopf and Heppner (1988). *The Counseling* Psychologist, *16*(4), 671-675. https://doi.org/10.1177/0011000088164009

Einhorn, S. (2012). Microworlds, computational thinking, and 21st century learning. *LCSI White Paper*.

Lister, R., Fone, W., McCartney, R., Seppälä, O., Adams, E. S., Hamer, J., Moström, J. E., Simon, B., Fitzgerald, S., Lindholm, M., Sanders, K., & Thomas, L. (2004). *A Multi-National Study of Reading and Tracing Skills in Novice Programmers*. 32.

Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J., & Szabo, C. (2018). Introductory Programming : A Systematic Literature Review. *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, 55–106. https://doi.org/10.1145/3293881.3295779

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I., & Wilusz, T. (2001). A Multi-national, Multi-institutional Study of Assessment of Programming Skills of First-year CS Students. *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, 125–180. https://doi.org/10.1145/572133.572137

Medeiros, R. P., Ramalho, G. L., & Falcão, T. P. (2018). A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Transactions on Education*, *99*, 1–14.

Muller, O., & Haberman, B. (2009). *A Course Dedicated to Developing Algorithmic Problem-Solving Skills–Design and Experiment*.

Nelson, N., Sarma, A., & van der Hoek, A. (2017). Towards an IDE to Support Programming as Problem-Solving. *Psychology of Programming Interest Group*, 15.

Oh, S.-H., Kim, E.-J., & Kim, S.-S. (2017). Development and Application of Educational Contents for Entry Programming to Improve Metacognition. *The Journal of Korean Association of Computer Education*, *20*(5), 61–68.

Sambe, G., & Basse, A. (2020). Ontology Based Framework For Learning Algorithm. *International Journal of Scientific and Technology Research*, *9*(01), 5.

Sprankle, M., & Hubbard, J. (2008). Problem *Solving and Programming Concepts* (8e éd.). Prentice Hall Press.