INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

The Java Learning Machine: A Learning Management System Dedicated To Computer Science Education

Martin Quinson Gérald Oster
Université de Nancy

N° 7537

2011

Domaine 3

ISSN 0249-6399 ISRN INRIA/RR--7537--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
NANCY - GRAND EST

The Java Learning Machine: A Learning Management System Dedicated To Computer Science Education

Martin Quinson Gérald Oster
Université de Nancy

Domaine : Réseaux, systèmes et services, calcul distribué
Équipe-Projet AlGorille

Rapport de recherche n° 7537 — 2011 — 11 pages

Abstract: This paper presents the Java Learning Machine (JLM), a platform dedicated to computer programming education.
This generic platform offers support to teachers for creating programming microworlds suitable to teaching courses. It features an integrated and graphical environment, providing a short feedback loop to students in order to improve the effectiveness of the autonomous learning process. This paper presents the motivations behind the platform and its main functionalities.

Key-words: CS1, Java, Education, Tools

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Java Learning Machine: une plate-forme pédagogique dédiée à l'enseignement de la programmation

Résumé : Ce rapport présente la Java Learning Machine (JLM), une plate-forme dédiée à l'enseignement de la programmation.

Cette plate-forme générique permet aux enseignants d'informatique de créer des micro-mondes utilisables dans leurs cours. Elle constitue un environnement graphique intégré, offrant aux apprenants d'obtenir un retour immédiat sur leur travail. Cela permet d'améliorer l'efficacité du processus d'apprentissage en autonomie.

Ce rapport présente les motivations ayant mené à la création de la plate-forme, ainsi que les principales fonctionnalités de l'outil.

Mots-clés : Initiation à l'informatique, Java, Enseignement, Outils

1 Introduction

In recent years, technologies to assist the collaboration and interactions between instructors and learners as well as between learners received a tremendous attention in education. The resulting tools, called Collaborative Management Systems (CMS) allow much more engaging education strategies, leading to more efficient learning paths. That is why most of the existing Learning Management Systems (LMS) providing computer-assisted teaching offer collaborative means.

When it comes down to education, computer science exhibits at least two specificities. First, the main referential is not the teacher, but the computer which acts as a definitive referee in debates. This is quite different to other disciplines where the teacher is much more in charge of defining the truth for the teaching. Another specificity of this science is that it is quite easy and cheap to setup an experimental lab since most of them only request a computer, while the needed equipment in other sciences such as biology or physic may be expensive and/or dangerous.

These specificities make auto-experimentation quite efficient and simple to setup, and allow some self-taught person to learn quite a lot in the domain. It is thus not a surprise that the constructivist psychological ideas found one of its main pedagogical applications in computer science through the constructionism learning theory with Seymour Papert's LOGO language [14].

In this paper, we introduce the Java Learning Machine (JLM), a generic infrastructure making it easy for teachers to create programming microworlds adapted to their teaching, and providing an integrated and graphical environment providing a short feedback loop to the students to improve the effectiveness of the autonomous learning process.

This paper is organized as follows: Section 2 motivates our work through a review of the state of the art regarding the use of microworlds to teach programming skills. Section 3 presents the didactic rational which guided us to build the platform. Section 4 presents how the tool can be used to both ease the teacher work and to increase the students motivation. Section 5 concludes the paper and gives some future work directions.

2 Programming Microworlds

The most successful aspect of the constructionism to teach programming was certainly the graphical Turtle of LOGO, although the original work entailed a comprehensive environment not centered on teaching programming. In this *microworld*, as it got called afterward, student controls an *actor* (a robotic turtle) interacting with its *environment* (a sheet on which the turtle leaves a trail as is moves). Several systems built upon the success of this system, such as LogoBlocks [2], which allows to construct programs visually by assembling blocks without the need of a textual programming language and eventually evolved into the well known Lego Mindstorm System, Roamer [7], in which the commands get executed by a real robot, StarLogo [17], to explore decentralized systems such as ant colonies, or MultiLogo [16], to experiment with concurrent programming.

It should be noted however that the interactions of the actor with its environment are rather limited since the turtle is "blind": it cannot check its microworld in any way. Several other systems were proposed in the years to solve this. The most popular is certainly *Karel the robot* [15], where the student control a robot using a language close to the pascal programming language. The actor, robot Karel, can move performs

tasks in a world that consists of intersecting streets and avenues, walls and beepers. The robot can carry beepers, and is provided with several predicates to check the state of his world, such as testing the presence of nearby walls or beepers. Several extensions of this metaphor were proposed over the years, such as Karel 3D [11] offering a richer interaction set, objectKarel [21] allowing the student to experiment with the Object Oriented Programming paradigm, or Kara [10] leveraging finite state machines to organize procedures. More comprehensive reviews of existing programming microworlds can be found in [5, 13].

3 Didactic Rational of the JLM

3.1 Applicability to Self-Learning

JLM was designed for the first course of our curricula, where absolute beginners are mixed to students who already learned the bases of programming elsewhere. That is why one of our first objectives was to allow learners to work at their own pace during the practicals, without waiting for the correction of exercises from teachers. For that, we integrated the evaluation of solutions into the environment so that learners can check by themselves whether they passed the exercise or not. Since the platform source code is freely available, this is not intended for grading purpose, but mainly to give students a better view of their level and what they should work onto.

Another advantage of this embedded auto-evaluation of the solutions is that puts the emphasis on the program action and not on its syntax. Indeed, when given a correction, some students discard their own solution because of its syntactic differences with the provided solution. In JLM instead, we never show corrections but help the students test whether their own solution is correct or not by comparing the *execution effects* of the program to what it was supposed to do. Hopefully, this helps students understanding that it never exist a unique correct solution to any problem. An interesting side effect of integrating auto-evaluation into micro-worlds is that one can easily provide a graphical interface featuring a very short interaction loop. The resulting learning infrastructure should be usable and motivating both for students with absolutely no prior knowledge in the field and for students with basic to advanced abilities in programming.

One of the limitation we see in most of the proposed systems is that they do only provide a limited environment to the students. In [20] for example, the microworld is mainly Java class, directly edited by the student in an external tool. The lesson objective is carried to the student through a printed or online handout. Following the example of tools such as [10, 21], JLM proposes a full learning environment, where the e-lessons are displayed, the source code is typed, and the execution is depicted directly. This also allows short interaction loops between the students and their environment and simplifies the execution of the environment by the students outside of the tutored sessions.

In order to replace the advice that the students may get from the tutor during in-class session, the exercises can contain textual *hints*, hidden by default and that the students can ask to see if they feel stranded with the exercise. This allows to build progressive exercises that the student are encouraged to solve by themselves, but containing helping elements to ensure that every students can achieve the assignments.

Since JLM was originally designed for in-class sessions, the support for computer-aided collaborative solution remain somehow limited at this point. In future work, an integration of the JLM as exercise runner in Moodle or the integration of forums

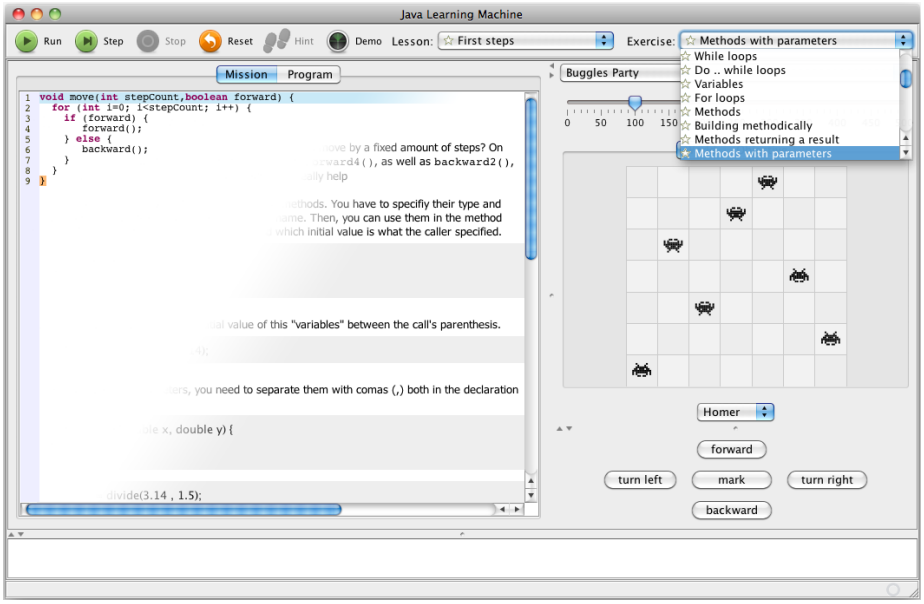

Figure 1: Main view of the JLM Platform.

allowing students to discuss could easily be envisioned to add a collaborative side to our tool.

3.2 Generic Infrastructure for ILOs

One of the limitations of existing micro-world solutions is that each of them only provide one microworld metaphor, where differing micro-worlds could be used to introduce differing programming concepts. In [20], the authors introduce three microworlds: the *BuggleWorld*, a generic microworld inspired from Karel the Robot; the *TurtleWorld*, inspired from Logo turtles used to introduce recursion through the drawing of polygons, spirals and trees; the *PictureWorld*, a graphics microworld inspired by the Escher picture language used in [1], allowing to construct complex pictures by transforming and combining simple shapes to exemplify recursion through the construction of quilt patterns. In [6], the authors introduce a large amount of microworlds, each aiming at reinforcing a specific programming ability. Unfortunately, in each case, every microworld were developed independently, inducing an extra burden to the teacher when building new metaphors to help their teaching. This burden becomes even more intractable when the amount of non-functional requirements increases, such as student code saving and restoring, program animation, step-by-step execution, automated solution grading, and so on.

On the other end, the use of generic multimedia scripting systems was proposed to introduce programming to students. Scratch [18] is such an interactive environment allowing kids to develop interactive stories and animations, where scripts are constituted of building blocks assembled visually. The Alice [8] environment was designed as a teaching tool to let students learn about object-oriented programming through the design of scripted 3D animations. These systems however do not provide support for teachers wanting to setup teaching situations through guided exercises where the student is given

an initial situation and must let the actors act onto the world to reach a wanted situation, similar to the Interactive Learning Objects used in [6] or [20].

In contrast, JLM provides a generic infrastructure allowing to easily add new kind of microworlds. Thanks to this infrastructure, and since the non-functional code is shared between the microworlds, the teachers can easily add new kind of microworlds adapted to their teachings. As an example, the world we wrote to teach the Hanoi tower problems lasts about 200 lines of code.

Another limitation often encountered in microworld solutions from the literature is that most of them use specific teaching languages (see [18, 8] for example), designed to be easier to understand and master than general-purpose programming languages such as C, C++ or Java. Even if this naturally eases the teaching experience, this introduces a specific difficulty when the student transition to the general-purpose language since the syntax switch can disturb them. That is why several microworlds sequels allow the use of a general-purpose language. For example, Karel++ [3] allows to control a Karel with C++ while Karel J Robot [4] permits to do the same in Java. Likewise, the upcoming version 3 of the Alice environment will allow the animation of 3D scenes in Java.

In JLM, most of the exercises are written in Java, but the infrastructure allows to construct worlds using other languages as well. As an example, we propose a specific world called LightBot [9], inspired from a classical flash game. The student controls a robot through graphical orders without the need of writing textual orders. As future work, we plan to leverage the Java Scripting Infrastructure provided by the JSR 223 [12] to allow the use of other programming languages such as Python or LISP.

3.3 Scaffolding Feature

Several researches have assessed that one of the main difficulties faced by novice programmers is the *extended instruction set* of generic purpose programming languages [5]. Moreover, Java poses specific difficulties, as students are faced to advanced notions from day one because of the typical `public static void main(String[] args)` program entry point [19]. This naturally induces the wish to *hide* details to students for pedagogical reasons.

To that extend, JLM allows to use *templates in exercises*. Instead of asking students to write full classes from day one, or to drown them in provided classes to modify, they are asked to write simple code *chunks* that are then automatically injected in the exercise template before being executed. For instance, in the first series of exercises that basically modeled the LOGO environment students are able to use variables, loops and basic method calls (such as `forward()`, `turnLeft()`, etc.) without knowing that they are implementing the core code of the `run()` of a specific `Tortoise` class. They even do not know the notions of class and object. In following series of exercise dedicated to object-oriented programming, scaffolding allows to introduce progressively object-oriented concepts (method, class, then interacting classes, etc.) and their Java syntax. Scaffolding is also used in JLM in order to mask useless part of code: for instance in the series of exercise dedicated to comparative sorting algorithms, students are only asked to write algorithms using basic primitives (`swap(i,j)`, `compare(i,j)`, `elementCount()`) without requiring to deal with array declaration, initialization or any object-oriented concepts. This allows to focus students attentiveness to the main exercise goals.

JLM offers another, more traditional, support for scaffolding: *incremental exercises* allow students to automatically reuse the code written in a previous exercise and adapt

it to the needs of the current one. For example, in the maze escape lesson, students are first asked to come up with a simple algorithm before forcing them to adapt it to much more complicated mazes.

4 The JLM Environment

4.1 Using JLM: the student point of view

The JLM constitutes a comprehensive environment to help the students building and improving their programming skills. It proposes a set of practical exercises to students because we believe that the exercising is the only way to acquire programming skills. Exercises are not isolated, but grouped by thematic e-lesson (comparative sorting algorithms, recursion, etc.), providing a coherent progression on the topic. Each exercise constitutes a specific learning situation, where the student should instruct the actors of the presented world to change the situation from the presented initial conditions to the objective conditions (both being graphically depicted in the interface). It is served by a textual description, introducing both theoretical background and details about the specific exercise goals.

Each exercise can studied from different perspectives. First students can use the *demonstration mode* to execute the expected solution they have to code. They can execute this code at different speed rate and even step by step. In this way, they learn and understand by observing the expected behavior of what they have to code. Then, in most world, students can use the interactive controls (buttons) in order to solve the exercise. For instance, in the world inspired from LOGO, they can give orders to the tortoise by clicking buttons. Finally, students can use the code editor tab in order to type in their code. This code is compiled then executed. The computed result is compared to the expected results of the solution. At every moment, student have access to the visualization of the effect of the expected solution. In order to test the solution proposed by students, their code is executed on multiple instances of the same problem. In this way, it is possible to test boundary cases of the algorithm to be developed.

4.2 Extending JLM: the teacher point of view

Teacher can reuse as-is the JLM platform and the proposed series of exercises. They can also easily extend the platform, either by writing their own exercises in the existing worlds or by implementing new worlds.

In order to write a new exercise, teacher has to write an HTML file describing the exercise and few Java classes (generally one) which extends the existing entities of the world chosen for this lesson. The entity to implement contains the expected solution that will be run in the demonstration mode. By including specific annotations in the source code, teacher configure the templating engine in order to set which part of the code has to be masked to students (because it is useless for the main goal of the lesson), which part has to be hidden (because it is the solution) and so on.

Extending JLM with a new world is the way to propose a new series of lessons with its specific visualization, entities and interactive controller. In order to achieve this, teachers have to implement few Java classes relying on the core framework of the JLM framework. Generally, it is one class for the world model (containing business logic and data), one class providing the graphical view for the world, and one class representing the abstract entity that will be derived to build solution and students proposals.

4.3 Technical Description

JLM platform is developed in Java and is freely available at `http://www.assembla.com/spaces/JLM/`.

Java was naturally chosen as the implementation language for the platform since it is the targeted language to be learned by our students. And, it offers several advantages.
Platform independent. It allows to develop an application which is independent of the operating system. JLM runs on most modern platform such as Windows, Linux or MacOS.
Easy deployment. Using the Java Web Start framework, JLM can be run without any installation, users have only to access a web page with their standard web browser. This allows transparent deployment of updates and new lessons.
Embedded compiler. Java API provides access to the embedded java compiler directly from the language. Thus, in JLM, when students write code and try to execute it, this code is first translated or decorated with templates to produce standard Java code which is compiled on-the-fly. Then, this compiled code is executed and results are compared to expected results of the exercise. Even if JLM compiles Java source code, it can be used to teach other languages. It is easy to develop domain specific languages for lessons or used any existing scripting languages available for Java (such as Jython).
Internationalization support. As lessons and exercises descriptions are standard HTML files, it is very easy to provide a translated versions. Currently, the proposed lessons are translated to English and French, and other languages could easily be added. We use the po4a[1] framework to ensure that every translation remain up-to-date.

4.4 Proposed Microworlds

The purpose of this paper is to introduce a generic microworld solution allowing the teachers to easily implement programming microworlds. In order to evaluate this, the current section shortly presents the most preeminent microworlds implemented so far within the environment and how the functionalities proposed by the environment helps in this process. Most of these microworlds are not original, and do not constitute the main contribution of this paper.

Buggle World. This world was the first to be implemented in JLM. It relies on an original idea of Franklyn Turbak, at Wellesley College ([20]). It is full of Buggles, little animals understanding simple orders, and offers numerous possibilities of interaction with the world: taking or dropping objects, painting the ground, hitting walls, etc. This world is used in two series of exercises. In the first one, students learn the basic concept of programming (variables, loops, conditionals, functions, etc.) and also different strategies (divide-to-conquer, etc). Contrary to the original version, students are not obliged to modify/complete existing classes and compiles all the required classes. They only have to write the code which is "interesting" for the exercise. Templating easily allows to hide the useless but inherent complexity of the system.

As an example of exercise, students have to write a piece of code that let buggles read letters on the ground, these letters being translated to orders (forward, turn left, etc.). The whole sequence of actions should make the buggles follows a specific dance choreography.

Another series of exercises use this world to teach students classical labyrinth algorithms (random, wall-follower, pledge algorithm, shortest path, etc.) to escape

[1] The po4a framework: `http://po4a.alioth.debian.org/`

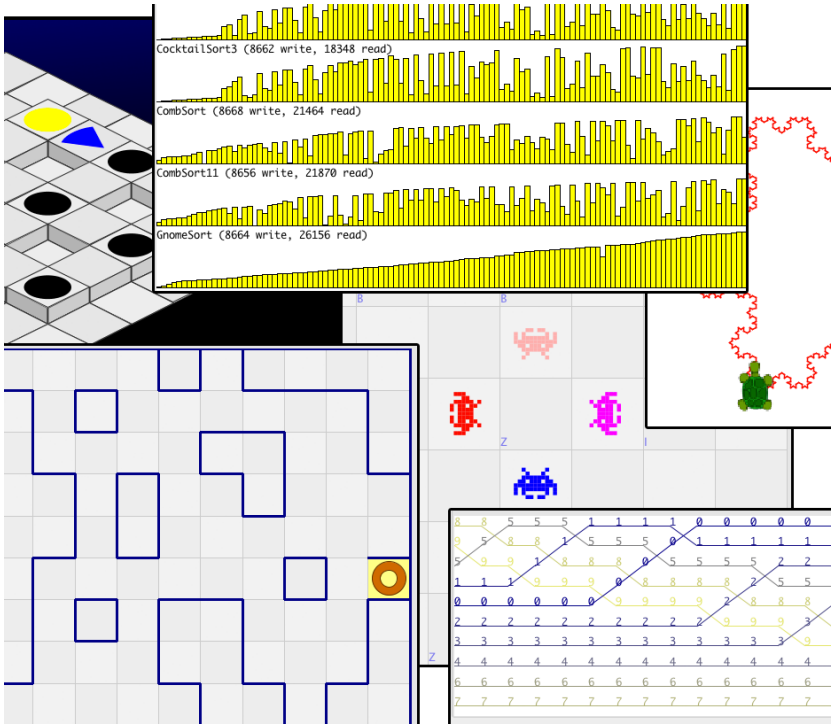

Figure 2: Visualizations of different existing micro-worlds (from left to right and top to bottom): LightBot, Sorting world, Turtle world for Recursivity, Buggle world for labyrinths and initiation exercises, and another visualization of the sorting world.

different kind of maze. Students have to implement the algorithm in order to make their player escapes the maze. A snapshot of this lesson is presented in the bottom left part of Figure 2.

Sorting World. This world provides tools to let students experiment with sorting algorithms. It can be used in different ways: the first one is naturally to write the required sorting algorithms. Students can also simply use the demonstration mode of each exercise to observe the behavior of sorting algorithms and get a practical idea of their time complexity. It helps understanding the differences between each of them since algorithms visualizations are provided side by side. The demonstration mode can also be used in order to execute step by step an algorithm and in this way understand how it works.

Turtle World. For studying recursive algorithms, we implemented a clone of the classical LOGO environment allowing the students to control turtles. The main difference is that students calls Java methods instead of using standard LOGO primitives. Each exercise presents a new recursive figure to draw. Students have to implement the underlying recursive function which is then executed with different values assigned to its parameter.

LightBot World. This world is merely a programmer puzzle rather than a real lesson (although some use it to teach programming). Students have to control a robot by giving it orders (forward, jump, turn-left, turn-right, light up, call function) in order to make it

light up all the lights located on the board. It use a programmer-style logic for solving complex levels that includes functions to-reuse orders. Students do not program their robot moves in Java, but rather graphically. The difficulty is that students are allowed to use a restricted number of slots (12 slots for main program, and 8 slots for each of two available functions) in which they put their orders.

5 Conclusions and Future Work

In this paper, we described our original work on Java Learning Machine, a platform dedicated to computer programming education. This platform provides an integrated environment that student can use to learn programming (not only Java) by interacting with microworlds. Students can observe the expected behavior of the solution of the exercise to be solved, then interact with the specific situation of the microworld and its entities, then program their own proposal to solve the exercise. The proposed code is executed and its effect is compared to the expected effect of the real solution.

As future work, we are currently collecting observations on how our students are using this platform. We envision to add tools that enables teacher to track the progress of each student in order to quickly detect whom of them are stuck. We are also considering the implementation of versioning mechanisms that will keep all versions of the code developed by students. This will allow both teacher and student to consult all trials made by a student to achieve a solution to an exercise.

References

[1] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs*. McGraw-Hill Book Company, 1995.

[2] Andrew Begel. Logoblocks: A graphical programming language for interacting with the world. Master's thesis, Massachusetts Institute of Technology, 1996.

[3] Joseph Bergin, Marc Stehlik, Jim Roberts, and Richard Pattis. *Karel++: A Gentle Introduction to the Art of Object-Oriented Programming*. John Wiley and Sons, New Jersey, 1997.

[4] Joseph Bergin, Mark Stehlik, Jim Roberts, and Rich Pattis. Karel J. Robot: A Gentle Introduction to the Art of Object-Oriented Programming in Java. Available at `http://csis.pace.edu/~bergin/KarelJava2ed/Karel++JavaEdition.html`, 2001.

[5] Peter Brusilovsky, Eduardo Calabrese, Jozef Hvorecky, and Philip Miller. Mini-languages: A way to learn programming principles. *Education and Information Technologies*, 2(1):65–83, 1997.

[6] Nadya Calderon, Jorge Villalobos, and Camilo Jimenez. Developing programming skills by using interactive learning objects. In *14th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'09)*, pages 151–155, Paris, France., July 2009. ACM.

[7] Dave Catlin. The roamer robot, 1989. Valiant Technologies.

[8] Matthew Conway, Steve Audia, Tommy Burnette, Dennis Cosgrove, and Kevin Christiansen. Alice: lessons learned from building a 3d system for novices. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 486–493, New York, NY, USA, 2000. ACM.

[9] Coolio-Niato. Light-Bot. Available at `http://armorgames.com/play/2205/light-bot`.

[10] Werner Hartmann, Jurg Nievergelt, and Raimond Reichert. Kara, finite state machines, and the case for programming as part of general education. In *Proceedings of IEEE Symposium on Human-Centric Computing Languages and Environments*, page 135, Stresa, Italy, 2001. IEEE Computer Society.

[11] Jozef Hvorecky. Karel the Robot for PC. In P. Brusilovsky and V. Stefanuk, editors, *Proceedings of the East-West Conference on Emerging Computer Technologies in Education*, pages 157–160, Moscow, April 1992.

[12] Java Community. JSR 233: Scripting for the java platform. Available at `http://jcp.org/en/jsr/detail?id=223`.

[13] Caitlin Kelleher and Randy Pausch. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Survey*, 37(2):83–137, 2005.

[14] Seymour Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, 1980.

[15] Richard Pattis. *Karel the Robot: A Gentle Introduction to the Art of Programming with Pascal*. John Wiley and Sons, London, 1981.

[16] Mitchel Resnick. Multilogo: A study of children and concurrent programming. *Interactive Learning Environments*, 1(3):153–170, 1990.

[17] Mitchel Resnick. StarLogo: an environment for decentralized modeling and decentralized thinking. In *Proceedings of the 1996 Conference Companion on Human Factors in Computing Systems*, 1996.

[18] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. Scratch: Programming for all. *Communications of the ACM*, 52(11):60–67, nov 2009.

[19] Eric Roberts, Kim Bruce, Kim Cutler, James Cross, Scott Grissom, Karl Klee, Susan Rodger, Fran Trees, Ian Utting, and Frank Yellin. The acm java task force project rationale. Technical report, ACM, 2006.

[20] Franklyn Turbak, Constance Royden, Jennifer Stephan, and Jean Herbst. Teaching recursion before iteration in CS1. *Journal of Computing in Small Colleges*, 14(4):86–101, may 1999.

[21] Stelios Xinogalos, Maya Satratzemi, and Vassilios Dagdilelis. An introduction to object-oriented programming with a didactic microworld: objectKarel. *Computers & Education*, 47(2):148 – 171, 2006.

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399