Olympiads in Informatics, 2012, Vol. 6, 69–85

A Self-Paced Learning Platform to Teach Programming and Algorithms

Mathias HIRON, Loïc FÉVRIER
France-IOI Organization
www.france-ioi.org
e-mail: mathias.hiron@gmail.com, loic.fevrier@france-ioi.org

Abstract. Self-paced learning platforms have received a lot of attention recently, with the promise of fundamentally changing education. For more than 10 years, the members of the France-IOI non-profit organization have been developing such a platform for teaching programming and algorithmics to self-motivated users ranging from beginners to advanced programmers. With the introduction of algorithmics in the French mathematics curriculum of high school, it is now starting to be used by teachers in classrooms. In this paper, we present and explain the choices we made when creating the contents and tools that constitute this platform.

Key words: self-paced learning, programming, algorithmics, exercises, e-learning platform.

1. Self-Paced Learning

Providing an education perfectly tailored to the needs of each student in a classroom has long been a dream for teachers as well as students. The practical constraints of giving an education to the vast majority of children in a country, while trying to give an equal chance to every child, can often make teaching and learning a very frustrating endeavor. Teaching children by batches of 25 or sometimes much more, with the criteria for forming the groups being mostly based on age and place of residence, makes it very hard for teachers to provide an effective educational experience to every child in the group.

Even if we could somehow form ideal groups where every student had the same background, the same level at the beginning of the year, the same set of skills and learning style, there would still be no way of giving a lecture in front of this group so that the pace and content of that lecture would fit every student's needs. Anyone can get distracted at some point and miss a key concept that is essential to understanding the rest of the lecture, and students who understood it well the first time will get bored and miss other parts if the teacher repeats it one more time to make sure everyone gets it.

With a self-paced approach to teaching, each student learns at her own speed, so at any time, every student in a group could be studying something different. The teacher is no longer the main provider of knowledge, the time usually spent broadcasting a lecture to the whole group is freed, and the teacher becomes available to guide students and help them individually. For a self-paced approach to work well, the external source of

educational content must be of a very good quality, very clear and engaging, so that each student stays focused and continues learning at a good pace while the teacher is helping others.

This need for a high quality source of educational content adapted to self-paced learning is one of the main reasons this approach has rarely been implemented in classrooms until recently. Books are not interactive enough, or engaging enough to be suitable for self-paced learning. New technology has long been expected to be the key, with videos or educational software promising big changes, but while the technology has already been available for many years, progress has been slow until recently. The lack of high quality resources made it hard for teachers to adopt this new style, and the lack of teachers using the self-paced model reduces the incentive for content creators to build the resources that are needed.

The trend is now changing, and changing fast. A single educational resource, initially created by one dedicated man, Salman Khan, who authored around 3000 short educational videos and put them on YouTube, seems to have been the key resource that was needed for self-paced learning to become possible and popular (Khan 2011). With millions of students now using the Khan Academy to learn at their own pace, this non-profit with now 30 full-time employees, has become the source of inspiration for several other significant initiatives in this field. One such example is Udacity, an online university co-founded by Sebastian Thrun, a former Stanford teacher, who after an experiment of successfully providing a Stanford course on Artificial Intelligence as an online interactive course to tens of thousands of students around the world, realized that he could have a much broader educational impact by creating online educational resources available to everyone, as opposed to teaching smaller groups of Stanford students in classrooms every year (Eddy, 2012).

As many other educators, the members of the France-IOI non-profit organization have been convinced by the potential of self-paced learning for many years, and worked hard on creating suitable educational content and teach programming and algorithmics to thousands of students. We learned a lot along the way about what makes an online resource effective for self-paced learning, and would like to share what we have achieved so far, and explain the reasons behind our choices.

2. Fast-Paced, Mastery Based Learning

2.1. *Mastery Based Learning*

One of the main issues with the traditional lecture-based teaching approach is that students move on from learning one concept to learning the next not when they are ready to do so and have mastered the previous concept, but when the teacher decides that it's time to move on. This time might be right for a small subset of her students, but is either too early or too late for the rest of the class (Gomes and Mendes, 2007). When the pace of learning is imposed, students quickly accumulate gaps in their knowledge of the domain that is being taught, and keep piling up new concepts on unstable foundations. Fast

learners are also victims of this approach, not only because they could probably move on faster and feel like they are wasting time, but also because no matter how good they are, there are always times when they get distracted or bored and miss some key part of the lesson.

The key idea of an effective self-paced learning approach is that a student doesn't move on to the next concept when the teacher tells him to, or even when the student feels like it, but simply when he has mastered a concept well enough to be ready for the next one. Therefore, an essential part of a good self-paced learning platform is a system that can determine whether a student is proficient enough with a given concept, so that he's ready to move on and use that concept as a stable foundation to keep building up his mastery of the domain.

Automatically assessing the proficiency of a student in a given concept is not an easy task, and the most common way to do that is to use multiple choice questions, which is far from being ideal. In the field of programming and algorithmics however, we have a much more effective way to automatically assess a student's proficiency: ask the student to write a program that involves the concept in question, and automatically grade that program against a series of tests, in the way it's being done in many programming competitions (Forišek, 2006).

The evaluation of programs is at the core of our platform. For each new concept, students are asked to write programs that read some input and provide their output either by using standard I/Os or through a library. The program is then evaluated on our server, where it's compiled and run in a sandboxed environment, against a series of test cases, within certain limits of time and memory. We use one that is part of the Moe grading system (Mares, 2009), which is the sandbox that has been used at recent IOI (International Olympiad in Informatics) competitions. Using this evaluation system, we built a comprehensive set of learning material, where each student learns a concept, practices it on exercises while proving his proficiency, then moves on to the next concept.

2.2. *Short Courses Motivated by Exercises*

When giving a lecture, there is a strong incentive for the teacher to present several new concepts in a row, or to go through a variety of examples of applications of these concepts during the same broadcasting session, before letting the students practice them on some exercises. Lectures often last an hour or more, without any activity on the part of the student, other than taking notes. Most students are not able to stay focused and fully engaged on the lecture for one or more hours at a time and often lose track at some point, not being able to connect again because they missed some of the earlier concepts.

The reason for this tendency is simply that it isn't very convenient to present one concept to the whole class in a couple of minutes, let everyone practice and wait for the majority to finish a couple of exercises to make sure they master that concept, before moving on to a new concept. Too much time is wasted switching back and forth from a broadcast mode to practice sessions, and it is much more comfortable to spend half of the time lecturing, and the rest practicing, or even just let students practice at home or during dedicated practice sessions.

With a self-paced learning platform and the knowledge being provided not by the teacher but by an external source, this incentive doesn't exist, and the cost of switching from learning a new concept to practicing mostly disappears. We see no more reason to group the presentation of several concepts together, before switching to practicing on this batch of concepts.

Each presentation should be short enough, that there is no time for the student to get distracted and miss part of what is being taught. Furthermore, if the student knows that she will need to apply the concept right away, it gives her a much stronger incentive to pay attention right now, and not wait until she really needs it for an exercise or the exam.

We go even further, by providing the reason for learning the concept before even presenting the concept itself: instead of providing a short course followed by an application exercise, we start by describing the exercise and follow on the same page with a short course that introduces the new concept needed to solve that exercise. The student reads the short course not because it is pushed to her, but because she is looking for a way to solve the exercise. Instead of being a passive receiver of a lecture, the student becomes active and seeks the information.

Typically, a content item on our platform is a combination of an exercise, and a very short course introducing the new concept needed to solve that exercise. This is the format we use for programming concepts that a student can't possibly discover by herself, typically a new element of the syntax of a programming language. When we teach more advanced concepts such as classical algorithms, we switch to an approach detailed in (Charguéraud and Hiron, 2008), where we give the student a chance to discover the new concept by himself, and provide the course in the form of optional hints, and detailed solutions.

2.3. *Many Small Exercises*

Our experience with traditional lectures in computer science is that fundamental programming concepts such as the notions of instructions, variables and affectations, conditions and loops, as well as the corresponding syntax, are taught in a very short amount of time. As an extreme example of this tendency, one of the authors witnessed a 4 hours lecture where all of these concepts and more were taught at once, before the students got any chance to write their first program. It is common to see students study advanced concepts such as object oriented programming and inheritance, while they are still uncomfortable with writing programs involving basic loops.

The issue is that there is not a lot that can be said about a concept such as the principle of the conditional instruction, without being tempted to immediately add more advanced concepts like boolean operators. The concept itself is very easily defined and explained, but mastering it is a very different story. Learning to program means learning a new way to think, and learning how to build all kinds of things using a very small number of building blocks. Even though a student can seem to understand and even be able to explain such a fundamental concept as the conditional instruction after a few minutes of introduction, it takes a lot of practice before he can really be considered to be proficient with that concept.

As a consequence, our introductory content is composed of many small exercises involving all types of uses of every programming concept, as well as many combinations of these concepts. At the time of writing, our platform offers no less than 18 exercises to practice basic if-else conditional instructions, and their combination with other basic concepts taught earlier. Continuing with the concept of boolean operators used within a conditional instruction means solving 10 more exercises.

Students learning on our platform write close to 80 small programs before starting to learn about concepts such as arrays and functions. By that time, they are getting very familiar with the basic building blocks of programs. It takes no less to build strong enough foundations and be ready to learn to use more advanced concepts on top of them.

2.4. *Detailed Solutions as an Essential Part of the Course*

The fact that a student was able to solve an exercise does not necessarily mean he understands every aspect of his solution. Students may have written parts of that solution without being entirely sure of what they were doing, or may have been helped by a fellow student or a teacher. They often have doubts and questions about what they just did, even when their solution is correct.

We try to make sure these doubts and questions are resolved by giving a detailed solution for each exercise, that includes a reference solution and explanations on how and why it works. The solution often resembles what the student has written, which gives her confidence and motivates her to continue. The explanations reinforce what the student has learned, and alleviate any doubts she may have about what she just did.

By providing reference solutions that are as clear and elegant as possible, we teach good habits by example. We observed that after reading our reference solutions for each exercise they solve and the justification for our choices, students often pick up our style and don't form too many bad habits.

To make sure the students really understand what is going on when their programs run and don't form an incorrect or blurry model of the execution of their solution, we try to show that execution from different points of view. We use detailed illustrations to present a visual model of the execution flow or of the state of the variables at a given time of the execution.

To go even further, we created a "simulator" (see Fig. 1), a tool that is inspired by the debuggers that can be found in many IDEs, and that allow users to run programs step by step. The IDEs that include these debuggers tend to have an overwhelming number of features, and asking students to use them can be counter-productive. Our simulator was created with only one goal: helping students to build a good mental model of what is going on during the execution of a program.

Students can run the reference solution step by step on any input data they provide. At each step, the current line of code that is executed is highlighted, as well as any changes to variables. These are shown within a representation of the heap and the stack, so that students start forming a good model of how memory is used, even before anything is explained to them about it.

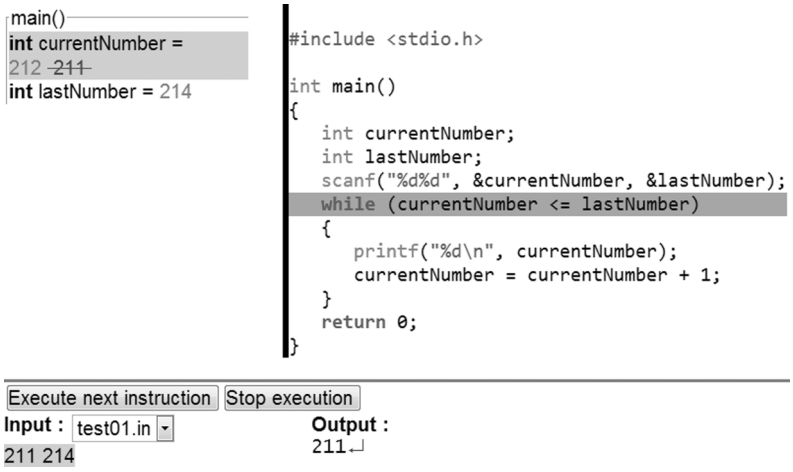

Fig. 1. Step by step execution simulator.

This simulator is far from being an actual debugger and can only be used on our reference solutions, but does a good job at what it is designed for. It is integrated directly on the solution page of the exercises, which makes it very convenient to use. Little by little, this tool helps students to learn to do mental simulations of the execution of their own programs.

Another ingredient of our solutions is the presence of reference solutions in other programming languages. Students are not expected to understand or study them in detail, but curious students who want to get a feeling of what each language looks like or students who have some previous experience with some of these languages can have a quick look by clicking on the corresponding tabs. This can help them to understand that the fundamental concepts of programming are independent from the syntax of the language they use.

As the exercises move from being purely about understanding the fundamental aspects of programming towards problems of algorithmic nature, the focus of our solutions becomes the thought process that can be applied from reading the problem statement to finding a working algorithm. For more information about this aspect of our approach, see (Charguéraud and Hiron, 2008).

3. Avoiding Frustrations

People have a natural tendency to enjoy learning, as it brings all kinds of good feelings: wonder, pride, self-confidence, self-respect and more. Learning is not always easy and the challenges are part of what makes people love it, but challenge often comes with some failures. Disappointment also happens when the material being taught is already known, and not much new is learned. Failing too often and for too long brings all sorts of bad feelings: frustration, disgust, low-confidence, shame, boredom, etc. Students learn well

when the good feelings associated with learning far overcome the occasional bad feelings. To be successful, a good teaching resource has to minimize these negative feelings while making sure students learn well and fast.

On top of creating good quality content and exercises, one of our priorities is to avoid the frustrations a student feels when getting stuck while trying to learn, either because he can't solve a given exercise, or because he doesn't know what to do next. Most of our users are using our platform from home, so when they get stuck and the platform doesn't help them, they are likely to just give up and never come back. Even if they are in a classroom, the teacher might not be available, or the student might just not feel like asking for help. We present several techniques that we employ to minimize the amount of frustration students will encounter while studying.

3.1. *Anticipating Errors*

Syntax errors are one of the main causes of frustration at the early stages of learning programming. Missing semicolons, parentheses, or double quotes are not obvious at all to beginners, and the error messages they generate often make things even harder to understand. While projects such as Scratch (Scratch, 2012) aim to avoid this issue completely by offering visual languages where no syntax error is possible, we believe that this approach only postpones learning to be careful and rigorous, which is an essential part of what programming is about. Execution errors such as infinite loops or divisions by zero can also be frustrating, and are not avoided by these tools.

We also noticed that a big part of the actual learning happens precisely when errors happen: as the student faces an error, all the details that they have doubts about are put into question, and he then actively tries to figure out the answer to his own questions. As a consequence, we don't eliminate errors completely, but reduce the risk that students get stuck for too long because of them.

The key idea of our approach is to anticipate errors, and teach students to recognize their own errors as well as the corresponding messages, before they even encounter them while solving an exercise. We do this in two different ways:

1. By providing courses that present different common errors associated with a recently learned programming concept, and show the corresponding error messages. Giving different examples and highlighting where the error is brings their attention to the types of errors that are possible. By showing and explaining the error messages associated with them, we reduce the amount of surprise and frustration the student might get in the future, if he encounters these same error messages. We don't necessarily expect students to remember each message, but merely recognizing something familiar and getting a rough idea of what types of errors can produce certain types of messages makes things much easier.
2. We give exercises whose only goal and difficulty is to find and fix syntax errors in one or more example programs. By solving these exercises, students learn to look for and fix the most common syntax errors. They are then more likely to notice their own errors in the future.

3.2. *Providing Help, through the Community or Automatically*

One way to make sure students never get stuck on a given exercise and always keep learning could be to give them access to the solution when they can't solve the exercise by themselves. Our concern with such an approach is that if it's too easy to get access to the solution, many students will be too tempted to read it before trying hard enough to solve the exercise by themselves. Some might even believe that they can save time and learn faster, if instead of really solving each exercise entirely, they only think about how they would do it, then look at the solution and see if they had the right idea. This might give them the illusion of learning for a while, but this learning will only be superficial.

Instead of giving them the solution, our goal is to give them the minimum amount of help that they need to continue by themselves. When possible, we avoid giving them even part of the solution and try to tell them something that will help them find the solution by themselves.

In the past, students who needed help contacted us by e-mail and sent us their source codes or ideas. The advice we gave them was often the same for a given exercise, so we created a system that provides these hints automatically, at the request of the students. Depending on the reason why students get stuck on a given exercise, these hints can help out very well, or be of no help at all, but this saved us and the students a lot of time, as they only needed to contact us in the latter case.

More recently, as the number of our users increased, and our free time diminished, we decided to set up a system so that other users could help each other. The principle of this system is quite basic: when stuck on an exercise, a student can ask for help on a forum where any student who has already solved that exercise can see the question and provide an answer. We try to make sure people who help others have access to all of the necessary information, by showing them the history of submissions of the student on that specific exercise (see Fig. 2). We make it very clear that helping a student doesn't mean giving him the solution directly, but only hints that make sure they don't stay stuck too long.

3.3. *A Clear and Well Designed Learning Path*

Making sure each concept is well explained and providing help to students when they get stuck is important, but there is another aspect that is essential to self-paced learning and that is too often neglected: making sure students always know what to do next, and making sure that what is offered to them corresponds to their level.

Online educational resources too often take the form of a multitude of courses and exercises accumulated in a big online database. With all these choices offered to them, students are expected to be able to find what they need. The paradox of choice (Schwartz 2004) showed that too much choice can generate anxiety for shoppers in a store, and the same is true for a student faced with a multitude of learning materials. Although we believe a student should have some choice for what to learn next, as she might be more in the mood to study one aspect of the domain rather than another, this choice should be limited to a very small number of clearly identified options, so that the student never feels lost and overwhelmed with the multitude of things to learn.

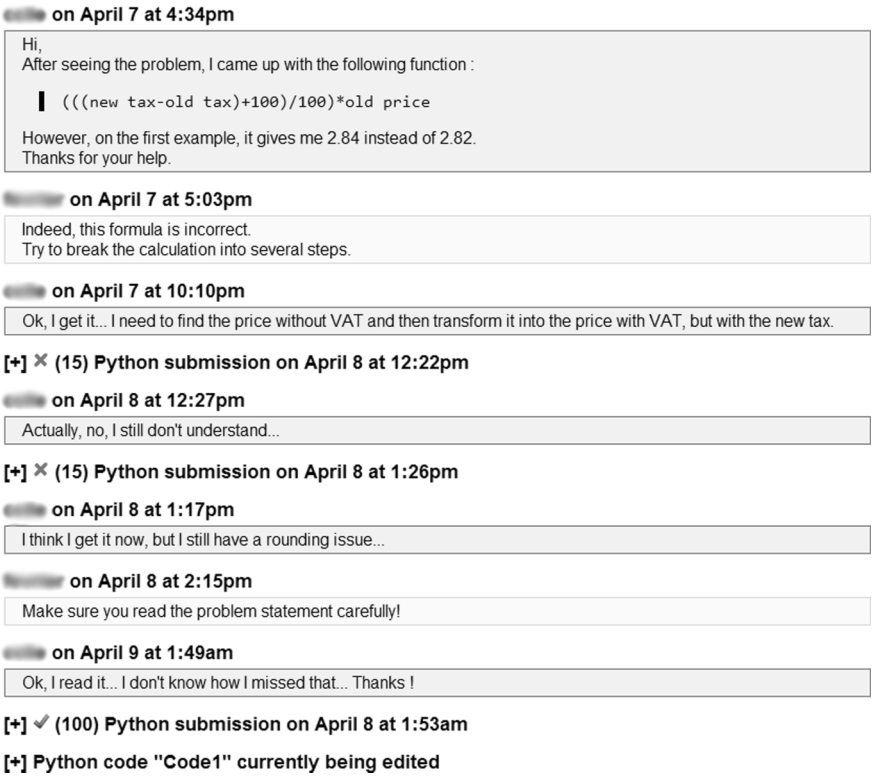

Fig. 2. Students helping each other on our forum.

The order in which different concepts are presented is an essential part of the quality of a curriculum. Not much is more frustrating to a learner, than studying material or trying to solve an exercise that assumes that some other concepts are mastered, if the student doesn't actually master them. Making sure the student is proficient in the prerequisites before letting her dive into some material is therefore key to avoiding this type of frustration.

On top of enforcing these constraints, we spend a lot of time thinking about the best order in which to present new concepts. Too many books or other introductory resources on programming focus on presenting the syntax of a language in a quasi-systematic way. As an example, after introducing integer variables to the student, they often continue with the other types (char, boolean, float, double, . . .). Learning these too early gets in the way of mastering the fundamental concepts associated with using variables. Our approach is to focus first on a small subset of the language features, that is needed to learn the fundamentals, and to keep additional elements of syntax for much later. We don't teach the syntax of a language, the language is only there as a means to teach programming, so we don't worry about making sure the student knows every aspect of that language.

This focus on teaching concepts in the right order is very apparent on our website. The content is split into a sequence of levels, and each level contains a sequence of num-

bered chapters, each focusing on a given theme. The courses and exercises in a chapter are also numbered, one of the many indications that students are expected to work on them in order. Students have the possibility to skip some exercises or chapters, as well as a whole level, but to skip a whole level and access to the next one directly, a student needs to solve a small set of challenging exercises to prove he is reasonably proficient with the concepts presented in the previous level. The first level starts at the very basics of programming, while at the time of writing, the last levels present relatively advanced algorithmic concepts, such as network flow algorithms.

4. Keeping Students Engaged

Learning can be fun, and has many other similarities with games. It can be easy at times and challenging at others. There is a sense of progress while playing a game and while learning, and both sometimes come with failures. In some cases, both can even become addictive.

Making learning more fun doesn't mean replacing some of the learning with games, or putting it inside a game. A better approach is to think about learning as already being some sort of game, and to try to make it a better game. For this, we can take inspiration from what works in successful games, and adapt it without changing the nature of learning. We have implemented a few ideas in this direction.

4.1. *Presenting Each Exercise in the Context of a Small Story*

Solving programming exercises can be fun even if the question is expressed in the most pure form. Writing a program that displays numbers from 10 to 0 can be interesting and motivating by itself, just for the motivation of successfully solving a task, and getting better at writing loops. However, it can be even more fun if the student does that while imagining that she is about to launch a rocket into space and needs to write the program that handles the countdown. When adding fun stories to exercises, the goal is not to hide the actual task and pretend that we're not learning, or even to justify that writing a program that counts from 10 to 0 can be useful, but rather to add to the motivation and diversify the stimulation associated with solving a problem.

The main drawback of putting exercises in the context of a story is that it means more text for students to read, which can be a serious obstacle for many students. When we write a fun story for an exercise, we try to keep the text as short as possible, and we make sure that students who don't like to read these stories or don't have the time to do so won't have to. We do this by clearly separating the story part from the rest of the exercise, and making sure nothing essential will be missed by students who skip the story. This is similar to games where some players like to skip videos and jump right to the action, while others really enjoy watching them.

To write interesting stories, we get some inspiration from various sources, going from Greek mythology to modern literature, scientific discoveries, or even noteworthy events of our history. We then use this opportunity to introduce students to the source of our

stories, and provide small excerpts describing the original context. For students who are interested in finding out more about the myth of Sisyphus, or "The man who planted trees" from Jean Giono, after solving an exercise that relates to these, we provide pointers to online resources about these subjects.

4.2. *Making Each Exercise Part of a Big Adventure*

Increasing the motivation and fun of solving each exercise is a first step, but to make sure students keep solving exercises and don't give up, we need to make them curious about what's next. A big part of it is simply to make the content interesting enough that students just want to learn more. Another dimension can be added in the form of a story that spans multiple exercises and chapters. The same characters can be present in different exercises, making each exercise part of a big adventure. Many aspects of a good adventure book or game can be applied, and we try to write the stories of our exercises in that spirit.

All of the exercises we created to introduce programming to beginners take place on an imaginary planet, with its own inhabitants, culture, plants and animals, and each story is part of a longer adventure. What we have done is far from being at the level of best-sellers, but students already enjoy it, and we can improve this aspect little by little.

4.3. *Recognizing Short Term and Long Term Accomplishments*

One key aspect that can make games and learning addictive, is the sense of accomplishment that one feels as he makes progress. This aspect of what motivates players has been clearly recognized in games, and sometimes pushed to a ridiculous level, where you get awarded all sorts of achievements for performing the most trivial tasks. Without going to such extremes, recognizing short term and long term accomplishments during the learning process can be a very good way to keep students motivated.

A very simple but surprisingly efficient way to achieve this is to clearly display a green check mark in front of every exercise, chapter or level that a student has fully solved (see Fig. 3). Students are proud to accumulate all these proofs of their accomplishments, and really enjoy having a full column of green marks rather than one with some gaps, or worse, red marks showing exercises that they have failed to solve yet.

A longer term recognition of the achievements of our users is the "level" that is associated with their user name. Users start at level 1, then become a "level 2" user once

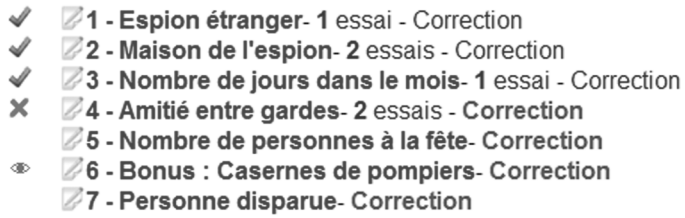

Fig. 3. Green check marks to recognize achievements.

they solved the great majority of exercises proposed in the first level of our content. We also work on measuring various parameters that we plan to associate with the "avatar" of each user: the experience (number of exercises solved), precision (based on the average number of attempts needed to solve exercises), persistence, and others. This is similar to characteristics that are associated with characters in role playing games, but in this case, these scores correspond to actual skills displayed by the student.

4.4. *Challenges*

While we teach the basics of programming to students who use our platform, we keep in mind that one of our goals is to teach algorithmics. The first few exercises that we use to teach each fundamental programming concept don't require to think much about what the algorithm should be, and are mostly focused on the implementation aspect. However, we often try to use exercises where some algorithmic thinking is needed, and more generally, problem solving aspects.

As an example, soon after introducing the concept of a sequence of instructions, we give students the following challenge: you are given a 5 liter and a 3 liter water containers. Using the instructions "fill(container)", "empty(container)" that fill or empty the given container completely, and "pourInto(source, destination)" that pours the content of the source container into the destination container until either the source is empty, or the destination is full. Using a sequence of these instructions, write a program such that at the end, the 5 liter container contains 4 liters. The solution is a sequence of 6 instructions, but takes some time to discover.

We noticed that even weak students tend to be interested in this type of exercises, and try really hard to find the solution by themselves. By solving this type of challenges once in awhile, we hope they will learn that if they try hard enough and long enough, they are able to solve problems that they had no idea how to solve initially, and without ever beeing told how to solve that type of question before. By giving them this kind of challenges very early, we try to show them how fun algorithmics and problem solving can be. We make the hardest of these challenges clearly optional, so that students don't feel too frustrated when they get stuck on one of them.

5. Content and Tools for High School Teachers, and Others

The very first version of our platform was created in 2001, as a tool to help us select and train students for the IOI. Computer Science was not taught as a subject in French schools at the time, and merely selecting the most talented self-taught French programmers was not sufficient for us to get good results. We decided to focus most of our efforts on developing this platform, and on training in general. This training platform had a very significant impact on our results, and our teams would from then on be awarded medals every year at the IOI.

In 2009, the French ministry of education decided to introduce algorithmic thinking as a part of the mathematics curriculum in high schools. By school year 2011–2012, it

was clear that most math teachers who were put in charge of teaching this subject had not found a good way to do so, having received no specific training to prepare them for this change. In August 2011, one of these teachers contacted us, to bring to our attention the fact that our platform had the potential to become a great tool for teachers to use in their classrooms. Until then, our target had mostly been self-motivated and passionate students, and even though we already had some programming courses used by thousands of users, these were not ready to be used by average high school students with no prior interest in the subject.

With the help of this teacher, Guillaume Le Blanc, who soon became an active member of our organization, we decided to dedicate a significant amount of time to creating content and tools specifically with high school students and teachers as our primary target. Although it was designed with a specific public in mind, we are convinced that these tools and content are suitable for anyone willing to learn programming, and any teacher who needs to teach programming to her students, at all kinds of levels.

5.1. *Ready to Learn Within a Few Minutes*

The first constraint a teacher is faced with before his students can start practicing programming on the classrooms computer, is to have the proper environment set up, with all the tools needed to edit, save, compile, and run programs. To make sure that installing such tools and teaching students how to use them doesn't take most of the first hour that the teacher can allocate to teaching this subject, we made sure to avoid this step entirely. Here is how the first programming class starts when using our platform:

- directly log into the computers; one student per computer is highly recommended,
- open a modern browser and go to france-ioi.org,
- create a new account, or log in using a facebook or google identity,
- click on the high school training section, and start working on the first exercise.

About 5 to 10 minutes after students enter the classroom, they have usually written and run their first program (the classical print "Hello World"), and are working on the second.

Programs are written in a very simple text editor directly integrated within the web platform. Programs written by students are automatically saved, compiled and run on our servers, and the results are shown in the browser, right below the editor. If a student has to leave while in the middle of an exercise, all he needs to do is to log out. He will be able to continue right where he was the next time he logs in. All of the time spent in the classroom is spent learning and practicing, and getting individual help from the teacher.

5.2. *Available Languages*

The official curriculum lets high school teachers free to use the language of their choice in their classes. We therefore offer our courses and exercises in some of the most popular programming languages among teachers. We started with Python because we and many teachers consider this language as being a good choice for beginners. We later adapted our

content to Java as well as Java's Cool, a simplified version of Java developed specifically for high school teachers by the INRIA (a French public research body fully dedicated to computational sciences).

Our exercises can also be solved using other languages available on our platform, such as C, C++, OCaml and Pascal, and we plan to adapt the associated courses for at least the first three of these languages. As our content is focusing on the fundamental concepts of programming, rather than on the details of the syntax of each language, the structure of the different versions is very similar.

5.3. *Adapted to the Official High School Curriculum*

The new official math curriculum (B.O., 2009) defines the programming concepts that every French student is expected to learn by the end of high school. One of the expressed goals is for students to be able to formalize some of the algorithms that they have already encountered, such as the Euclid algorithm (the computation of a GCD), and implement them using a programming language. Other algorithms such as statistical algorithms (mean, median, ...) or the binary search algorithm should also be taught.

To achieve these goals, the most fundamental programming concepts are needed, and students are expected to learn the concepts of sequential instructions, variables, basic calculations, input and output manipulations, as well as conditional statements and loops. Basic knowledge of arrays is not explicitly listed, but is implicitly required, as arrays are needed to implement most algorithms involving statistics.

The programming concepts listed in this official curriculum are no different than the first concepts any person interested in learning programming has to learn. As our users come from a wide range of backgrounds, we decided to make as few assumptions as possible on their proficiency with mathematical concepts. As a consequence, we have clearly separated our introductory exercises into two categories:

1. Exercises and courses dedicated but limited to introducing the programming concepts. They form a clearly identified linear progression that constitutes the first chapters of the 5 levels of our core content The chapters that correspond to the official curriculum contain around 100 small exercises, and an average student is expected to be able to solve most of these by spending about 20 hours of work.
2. Exercises and courses where these concepts are combined with mathematical concepts. These exercises are grouped by mathematical subject, and the algorithmic prerequisites are always given. For each subject, the underlying mathematical concepts are introduced and progressively mastered, from an algorithmic point of view. This category now has 40 exercises, and we are planning to greatly increase this number with the help of teachers, and hope to cover most of the mathematics curriculum in high school.

5.4. *Group Management Tools*

One of the great advantages of a self-paced learning platform is that, as students make their way through the set of exercises offered by the platform, data is collected about their

attempts, successes and errors. This data can be very useful to a teacher, who can get a good idea of which concepts each student has mastered, and which concepts they might need help with.

Our platform gives teachers the possibility to create groups that their students can then join. For each of the group he manages, the teacher has access to a summary of what exercises each student of the group has worked on or solved, at what time, and after how many attempts, and get access to the details of each submission. The current version of these tools is relatively basic and we have plans to improve it based on teachers' feedback.

5.5. *Early Feedback*

Our high-school ready content is available since the end of 2011, and several teachers have started using it as the primary way to teach programming and algorithmics to their students. The results are already very promising, and both the teachers and the students appear to appreciate this approach very much. A number of teachers are practicing on our platform, to get ready to use it for the next school years. The following is a testimony from one of the first teachers who have adopted our platform in their classroom:

I have started using the France-IOI platform to improve my own algorithmic knowledge and to complete the small formation I had received. I then used it both in the computer science club I animate, and with my regular classes, with small groups of 15 students at a rate of approximately one session every 2 to 3 weeks. I generally ask them to do at least one exercise each week at home, in order to maintain their knowledge.

Students love this way of learning, and this became their favorite activity. The overwhelming majority of my students are strongly engaged and study hard. They frequently ask me to do more of these sessions, but due to time constraints, I cannot satisfy their request.

There is no other activity for which I can obtain the same level of engagement from them, although I am not able to pinpoint the reasons for this. One thing that is certain, is that this is not purely due to working with computers. I organize other math-related activities on computers, but students appreciate them less than working with this platform, and sometimes would rather do them without computers.

6. Conclusion

In this paper, we present the features of the self-paced learning platform created by the members of the France-IOI organization and available on france-ioi.org, where anyone can learn programming and algorithmics from the beginning to an advanced level. We describe many of the choices we made when designing the current version of the tools and contents that are made available, and explain the motivations behind these choices.

We show how students learn the fundamental concepts of programming by solving many small exercises accompanied with very short courses, in a web-environment where they can directly edit, compile and run their programs. Following the principles of mastery-based learning, the solutions written by students are validated automatically

before they move to the next problem, and this system works with a variety of popular programming languages. The detailed solutions that are provided help them to reinforce their understanding of the concepts, and make use of a step by step simulator that helps students to form a mental model of how the computer executes a program.

We then explain how we try to avoid the frustration that often comes when faced with errors by anticipating them and explaining the most common ones, and how they can be recognized, and we describe how we prevent students from getting stuck by providing them automated hints, as well as a community based help system. We talk about our work on offering a very clear learning path to students, which we consider as being as important as the individual resources themselves.

We describe how we present each exercise in the context of a short story, with each of these stories being part of a big adventure that makes learning even more fun and engaging. We take inspiration from successful games while presenting this adventure, and motivate students by recognizing and clearly displaying their accomplishments. Easy exercises help students to quickly gain confidence, while harder exercises clearly identified as challenges make sure they practice their problem solving skills early.

We made sure that the first chapters of our content and the tools that come with them were perfectly adapted for the use by teachers in high school classrooms, in response to the introduction of notions of programming and algorithmics in the official high school math curriculum. We made sure students can start learning on this platform within minutes of entering the classroom, and can stop at any time without losing their work, and continue from home. With our group management tools, teachers who are freed from giving a lecture can follow what each student is doing and provide personalized help accordingly.

All this is work is done by a team of volunteers, including former IOI contestants and teachers. We do this because we are passionate about algorithmics and teaching it, and we believe all students have a tremendous potential that can they can reach if they have access to the right content and tools. As we don't want to restrict the access to any particular group, we are looking for volunteers to help us extend the reach of this platform, by translating existing content into as many languages as possible. The platform is ready for internationalization, and translation efforts have already started in various languages, such as English, Spanish, Lithuanian and German.

References

B.O. (Bulletin Officiel) (2009). Number 30, July 23. `http://media.education.gouv.fr/file/30/52/3/programme_mathematiques_seconde_65523.pdf`. Accessed March 2012.

Charguéraud, A., Hiron, M. (2008). Teaching algorithmics for informatics olympiads: The French method. *Olympiads in Informatics,* 2, 48–63

Eddy, M. (2012). Sebastian Thrun, Mastermind of Stanford's Free AI Course, Forms Free Education Website. `http://www.geekosystem.com/sebastian-thrun-udacity/`. Accessed March 2012.

Forišek, M. (2006). Security of programming contest systems. In: *Informatics in Secondary Schools, Evolution and Perspectives*. Vilnius, Lithuania.

Gomes, A., Mendes, A.J. (2007). Learning to program – difficulties and solutions. In: *Proceedings of the International Conference on Engineering Education*.

Khan, S. (2012). Let's use video to reinvent education TED talk.
`http://www.ted.com/talks/`. Accessed March 2012.
Mares, M. (2009). Moe – design of a modular grading system. In: *Olympiads in Informatics*, 3, 60–66.
Scratch (2012). `http://scratch.mit.edu/`. Accessed March 2012.
Schwartz, B. (2004). *The Paradox of Choice: Why More is Less*. ECCO, New York.

M. Hiron is the French IOI team leader, and is the cofounder and president of France-IOI, the organisation in charge of selecting and training the French team for the IOI. He creates tasks on a regular basis for national contests and online training programs, as well as for the French–Australian Regional Informatics Olympiad, and occasionally for other international contests such as the IOI or the APIO. He is a business owner working on projects ranging from web development to image processing and artificial intelligence.

L. Février is a PhD student in the Computer Science Department at the University Pierre and Marie Curie of Paris. Deputy leader of the French team at IOI 2011. His research interests focus on serious games applied to the learning of programming and algorithmics.