Nothing to fear but fear itself: introducing recursion in lower secondary schools

Violetta Lonati, Dario Malchiodi, Mattia Monga, Anna Morpurgo
Dip. di Informatica – Università degli Studi di Milano – Milan, Italy
{lonati, malchiodi, monga, morpurgo}@di.unimi.it

***Abstract*—Recursion is a powerful conceptual tool, nevertheless it is often considered too abstract and technical to be effectively proposed in lower secondary school. Here we present our experience in introducing 8th graders to recursive strategies where the active observation of recursive algorithms execution allowed pupils to understand the main features of a recursive process, to convince themselves that it works, and to unveil the mystery of its effectiveness. In fact, by focusing on the self-similarity of the input data and the possibility to delegate blindly the solution of subproblems, pupils were able to recognize the heuristic potential of recursion. We started with an unplugged activity using LEGO bricks where a recursive algorithm was executed by pupils, and we continued with an activity supported by a software tool we developed *ad-hoc*. The undertaking was concluded by the abstract consolidation of the basic concepts and properties which had come out during the previous activities.**

I. Introduction

> *Of all ideas I have introduced to children, recursion stands out as the one idea that is particularly able to evoke an excited response.* [Seymour Papert, 1993]

Since 2008, we organize activities aimed at showing to pupils the actual nature of informatics [1], [2], [3], [4], with an emphasis on its scientific aspects, often ignored in non vocational curricula in our country [5]. One of the major goals is to orient pupils in their further studies, and our activities are designed to convey the core of the discipline, focusing on representing and processing digital pieces of information.

Given its iconic status in the field, *recursion* seemed a wonderful theme for such activities. In fact, recursion is one of the most recurrent memes in computer science: folklore is full of jokes (even Google answers to a search of the term 'recursion' by asking "Did you mean: recursion"[1]) and ubiquitous references to chicken-egg or 'bootstrap'[2] techniques. But recursion is not just an alluring topic: if mastered it also provides a powerful approach to problem solving. Thus, one would expect recursion to be part of any introduction to *computational thinking*, but this is not the case, since it is often considered too "technical" or "advanced" to be presented to pupils, with the partial exception of vocational ones.

In fact, while there is a vast literature on teaching recursion (for an accurate recent survey refer to [6]), it usually focuses on older students (at least attending upper secondary school, but mainly at the bachelor level) and assumes a previous exposure to a programming language. Even the few devoted to middle schools have rather high prerequisites. For example, [7] studied pupils aged from 10 to 14 who qualified by achieving high scores in an IQ test; then they attended several courses in Logo lasting from 25 to 150 hours (75 hours on average). [8] focuses on recursion as a problem solving approach without explicitly referring to a programming language: it addresses students in Polish (upper) secondary schools, and the audience is assumed to be rather sophisticated, since it has probably already encountered (but possibly not mastered) recursion in primary or middle schools, using Logo or Pascal.

Our context is certainly much less mature, but still, we think recursion could be a proper topic. Indeed we believe recursion could be one of the best showcases of our discipline, with its mix of fascinating, powerful and challenging facets. Our first goal, however, was to convince pupils that a recursive strategy is nothing to fear of, that it works in many familiar cases and that its computational power, while fascinating, is not magic.

It is worth noting that in our country teachers for the selected age group are not trained in informatics and it is possible they do not even know what a recursive solution is: after all, this theme is not mentioned in the school curricula. So we did not design our activities as an alternate methodology to teach recursion to young pupils, but rather as a new proposal to stimulate further interest in informatics. We remark that our activities were not designed to learn recursive programming: devising a solution to a computational problem and *coding* it are separate concerns that require different skills and call for a different training [4]. Here we focused on the abstract problem-solving aspect of recursion, since we believe it is a trick that might attract pupils to the actual scientific essence of informatics, and being familiar with it could be both suitable and useful for a general audience.

Due to their extracurricular nature, our activities cannot exceed the time span of a workshop lasting a few hours. Thus, it was clearly out of reach that at the end pupils completely understand what recursion implies, or even how to use it in a new context. Rather, the goal we set was to give pupils a chance to familiarize with problem solving by *structural recursion*, and to put them in contact with the heuristic power of seeing a self-similarity in the structure of a problem: in these cases even a 'blind' delegation (*i.e.,* a delegation unaware of the ultimate end of the computation) may be used to describe the solution. Indeed, our preliminary

[1] As it often happens, Donald Knuth is more precise, even in jokes: The Art of Computer Programming, Volume I has these two index entries 'Circular definition, 260, *see* Definition, circular.' and 'Definition, circular, *see* Circular definition.'

[2] A word coined after the English idiom *'to pull oneself up by one's own* boot-straps*'*.

results support the hypothesis that basic recursive strategies are within the Vygotskij's proximal development zone [9] of 8th graders without a specific programming training. Pupils built up their own intuition that this strategy can indeed work in several cases and they got to see this technique as an effective approach for solving (generic) problems of different kinds just by exploiting the self-similarities of input data.

The paper is organized as follows: Sect. II briefly reviews the key aspects of recursion addressed by our activities; Sects. III and IV illustrate the goals, the methodology, and the workflow of the activities proposed in the workshop, while in Sect. V we present the assessment design and its results; the last Section is devoted to some concluding remarks.

II. Recursive solutions to computational problems

The first abstract step in approaching recursion is the notion of computational problem. A *problem* is specified by a function that associates a desirable output to any possible input data chosen from the problem domain. Computational problem specifications emerge every time one is able to frame a real life requirement as a data processing model. In several practical cases (*i.e.,* the class of *computable* problems), computational problems admit algorithmic solutions. It is worth noting that this notion of problem is somewhat different from the one most pupils have in lower secondary schools: often what they call a 'problem' (like "find the Greatest Common Divisor of 42 and 24") is in fact what a computer scientist would have probably framed as the 'instance of a computational problem' (like "find the GCD of two positive integer numbers"). Even in the first examples of programming shown to children the case of "family" of problems is often absent: the code implements an algorithm composed by unambiguous steps, but it works only for a specific input or a given problem environment (for example a specific maze). This is however a crucial step in grasping the idea of recursion, since its power relies on the reduction of a problem to a simpler one with the same "shape".

In fact, when input data naturally expose a *self-similar* structure (*i.e.,* their aggregate composition can be described in terms of themselves: for example, a sequence can be an empty one or made up by a first element added to another sequence) it is often convenient to devise a *recursive* solution (*structural recursion*), which builds the whole solution by combining the sub-solutions obtained by applying itself to "smaller" versions of the input data, or getting the result directly in the trivial cases. This pattern is so common and natural that several (functional) programming languages libraries factorize it in a 'fold' operator. It is worth noting that the process is in fact an exact analogue of mathematical induction and in general it simplifies correctness proofs (or, one might say, the general understanding about how it works). Less naturally but even more powerfully, it is sometimes possible to devise a recursive solution by generating new proper self-similar data *ad-hoc* for the problem at hand (*generative recursion*). We will focus on structural recursion, which is the most intuitive one and yet quite tricky to grasp, possibly due to the negative aura that self-references have in other fields, in particular in the context of definitions: pupils learn early that definitions must not use the *definiendum* in order to be acceptable in scientific discourses. Thus recursion is potentially perceived as a weird exception to an already well establish (and somewhat obvious!) rule. Obviously enough, at least for those who have already understood recursion or induction, the cornerstone to see recursion unexceptional with respect to the "no *definiendum* in *definiens*" rule is the *base case*, the case one is able to define trivially (yet sometimes difficult to get by the inexperienced [10]). Moreover, it is well known that those who misunderstood recursion often have mental models related to "loops" [7], [11]. The right model, instead, should be based on "copies" of the solution: our activities were designed to convey exactly this idea.

With the goal of presenting recursion as a heuristic means, our activities emphasize two key concepts: (1) **self-similarity:** recursion makes sense when data are composed by *smaller* versions of themselves and eventually there is a *smallest* version for which the solution is trivial; (2) **blind delegation:** the solution of a *smaller instance* can be delegated blindly, assuming that eventually the chain of delegations will delegate the smallest instance, which admits the trivial solution. Ingenuity is still needed to devise a proper way of combining the output of delegations, but the feasibility of the procedure should be now a direct consequence of the structure of the problem data. It should also be evident that the second concept is nothing more than a special case of a more general 'divide and conquer' strategy to which pupils are generally already somewhat acquainted: a difficult task can be sometimes made simpler by dividing it among a team of fellows. The novelty, however, is that now the sub-problems preserve the same structure of the main one.

All in all, we aim at promoting a grasp of the inductive nature of recursion, something [12] labels as the 'expert-level' abstraction of recursive thinking: however, we do not believe this will necessarily stem from a bottom-up understanding of how it works in programming languages.

III. Learning goals and methodology

Having in mind the school level of our target pupils (8th graders), we assumed some prerequisites:

- be able to execute a simple procedure described in natural language, even if recursive;
- be acquainted with 'divide and conquer' strategies, i.e., know the benefit of dividing a task into smaller tasks, executing them separately and them putting together their outcomes.

Being designed as extracurricular, the activities we are describing needed to have a short duration: two/four hours. Hence we do not expect pupils to develop strong specific operational skills; on the contrary the learning goals we aimed at are quite general or introductory:

- accept that the definition of an object may refer to the object itself, provided that a (trivial) base case is defined independently;
- understand that the self-similarity of the instances of a problem can be exploited to design a recursive strategy that solves it;

- be aware that a peculiar 'divide and conquer' strategy (which is called recursion) can be used to solve a problem when its subproblems share the same structure;
- be able to identify the fundamental features of a simple recursive algorithm, given its description or the possibility to observe its execution;
- be able to adapt a simple tail recursive algorithm to solve a new problem with self-similar instances.

In order to achieve these goals, we identified two crucial steps. First, one needs to understand that a family of similar specific problems can be set and dealt with as a single general problem, by searching for a general method that applies to each specific problem and allows to solve them all in the same way. This is characteristic of any computational approach, but it is especially relevant when recursion is considered, since each subproblem has to belong to the same family that contains also the initial problem to be addressed. However, such point of view is not at all usual at this age, hence the activities we proposed are designed also to increase pupils' awareness about it.

A more specific crucial step is understanding what happens during the execution of a recursive algorithm, in both getting aware of what happens at each single step of this process, and meanwhile grasping, at a higher level, how such steps are inter-related. Most activities we proposed focused exactly on this issue and in particular we decided to use computers to provide an environment that could foster this understanding. We thus developed an *ad-hoc* software tool that enabled pupils to make experiments about an already implemented recursive algorithm, in order to analyze and understand its behavior and features. Since we were interested in the features of recursion, rather than the implementation of recursive algorithms, the role we assigned to computers as in the approach already proposed for instance in our *Greedy money* learning unit [3] on greedy algorithms, was completely different from the one that can be seen in our *Mazes* learning unit [4] and, more in general, in proposals like *Code.org* [13], where the focus is on coding a program and running it in order to test whether it is able to provide a solution to a given problem instance.

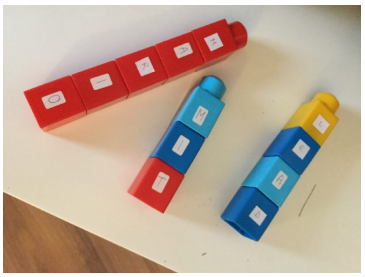
Common choices to introduce recursion are the Towers of Hanoi or fractal drawings. Considering the school level of our target pupils, we instead chose a simpler leading theme, namely manipulation of sequences of characters (*strings*). While very simple, strings are still interesting objects since they have a structure characterized by different forms of self-similarity: for instance a string can be pictured as the concatenation of a character and another string or, more generally, of two strings whose summed lengths equal the length of the original string. Also, several computational problems of various complexities can be proposed: *e.g.,* computing the length, reverting the letters, or sorting them.

From a methodological point of view, the activities were designed to foster the individual responsibility of pupils in their learning process. Hence, following the guidelines of experiential learning theory [14] and problem based learning [15], traditional explanations were avoided and replaced by group activities in an allosteric environment [16]. According to this approach, the teacher plays the role of a *facilitator* of the learning process, in that he/she has to provoke pupils with questions and stimuli, instead of instructing them directly. In particular, our ''algomotricity'' approach [17] aims at letting students build their mental models of the topic under investigation with "unplugged" and motoric activities (*i.e.,* LEGO bricks to represent letters of strings, see Sect. IV-A), followed by a computer-based activity.

Fig. 1. Pseudo code for the recursive function executed using LEGO bricks.

```
When the mate on your right asks to establish how long a word is,
follow these instructions:
if the word has one letter only
    whisper 1 to your mate.
else
    take off a letter from the word and put it in the trash;
    pass the rest of the word to the mate on your left;
    ask him/her to establish how long it is;
    wait for his/her answer;
    add 1 to his/her answer;
    whisper the result to the mate who had asked you.
```

Fig. 2. LEGO towers representing words



According to this methodological approach, the choice of the setting for the activities is critical in that it may determine how engaging the activities are for pupils. For the simulation of the execution of a recursive algorithm we took inspiration from the *The little people metaphor* [18], but we decided to change the figure of "specialist" with that of "fairy". This choice was aimed against the stereotype that problems in informatics are mainly solved by male characters [19], and was motivated also by the fact that fairy characters are more suitable to play on the fascination, the magic aura, of recursion.

IV. Description of the Activities

The workshop was structured in four steps, it involved eighteen pupils of one 8^{th} grade class, took place in the pupils' school and lasted three hours (two hours for the first two steps, and one hour the next day for the last two steps).

A. Computing the length of a word with LEGO bricks

Pupils were split into groups of five/six and each group sat in a row of seats. Each pupil received a note with the instructions he/she should carry out.

To start the activity, the rightmost pupil in each group was given a LEGO tower representing a word (each brick was labeled with a letter, see Figure 2) was asked how long the word was, and was told to apply the instructions written on the

note in order to answer. The note the pupils received had the description, written in natural language, of a recursive function computing the length of the word passed as argument, as in Figure 1.

We did not announce the final result the group was going to compute (the word length), otherwise pupils could be tempted to carry out the task in more natural and standard ways, and stressed that the instructions were to be executed literally, even though the process could appear mysterious at first.

A LEGO tower is a natural choice to give pupils a tangible example of a self-similar entity; it is easy to see the parts and also to assess the self-similarity of their composability: one can attach a single brick or a tower of bricks in the same way.

The problem considered (compute the length of a word) was very simple, but the algorithm was based on a novel (and initially strange) approach for the pupils with respect to their previous knowledge.

When every group had completed the task, other executions were performed, changing the positions of pupils within the group and the word to be examined, and using words with different lengths.

Pupils did not know in advance that they had all received the same instructions (notes were written with different styles) so that they were induced to pay attention to what was happening in the process; nevertheless, as we expected, they already noticed that the instructions were identical at the second repetition.

By changing words and positions, they were able to: observe the process from different perspectives, which in particular held for the leftmost and rightmost pupils, observe as many details as possible, focus on what they were doing both individually and collectively, observe the process on different instances of the same problem.

After a sufficient number of experiments, the facilitator led a debriefing discussion with the whole class on what had gone on.

B. Analysis of a recursive algorithm with a software tool

In the second step pupils were asked to analyze how a recursive algorithm works, with the help of a software tool developed for the purpose[3].

Pupils worked in pairs on the computers. The software tool allowed them to run the procedure (an animation of a recursive computation on a string) and to pause the execution, in order to observe characteristic aspects of the process—increasingly specific as they proceeded in the software 'levels', see below—and thus to capture the key features and follow how the algorithm worked. The "little people metaphor" and its visualization is ideal to suggest the "copies" mental model common in expert users of recursion [21], [7].

Although the algorithm was already implemented, the assignment here did not consist in a pure execution as in the first step, but it required a work of analysis and abstraction: using the revised Bloom's taxonomy [22], the former task is within the cognitive category of "understanding" while the latter is within the category of "analysis". Hence, to make the assignment feasible, we chose an algorithm similar in structure to the one of the previous step. Such algorithm is defined by a recursive function returning the reverse of the input string (see Figure 3).

Fig. 3. JavaScript code for the recursive function used by the tool.

```
function reverse(a_string) {
 if (a_string.length == 1) return a_string;
 else {
  var first_letter = a_string.charAt(0);
  var rem_string = a_string.slice(1);
  var reversed_rem_string = reverse(rem_string);
  var reversed_string = reversed_rem_string + first_letter;
  return reversed_string;
 }}
```

The activity was guided by a worksheet which, by means of several questions, accompanied the pupils in the discovery of the key components/features of the algorithm. The questions were intended to lead the pupils gradually through the disclosure of the process, allow them to observe concrete examples of how the algorithm worked, guide them in the design of their experiments, make them think on what they had observed so that their doubts could come up and they felt encouraged to make new experiments, while letting them make generalizations based on their observations.

The software tool lets pupils choose the string for the input and start its processing. After the execution, the interface shows the string obtained as output. The tool in particular displays a "computer" with a monitor which shows some objects moving on it, whose role is to represent the underlying process in some concrete way:

- a sequence of circles represents the recursion stack[4], each circle corresponding to a call of the recursive function, and there are as many circles as there are characters in the input string;
- the substring passed as argument of a call is represented by a yellow tower under the circle corresponding to the call and its height is proportional to the substring's length;
- in the first phase of the process, when calls are made, the tower moves from left to right and it shrinks, whereas in the second phase of the process, when values are returned and composed into the solution, the tower moves back to left and it grows;
- to keep track of the recursion phases, the sand in an hourglass changes its color: it is yellow in the first phase (stack of calls), orange in the second phase (returns), and red for the base case (when the argument has length 1);
- the body of the recursive function is executed by a fairy who can be seen perfoming the following actions: detach a piece from the top of the tower, pass on the tower, receive the tower, attach a piece at the bottom of the tower, wait, sleep.

[3]The software tool is a web application based on the Raphaël JavaScript library [20]. It is available here: http://aladdin.unimi.it/sw/fatine/ (in Italian).

[4]Although a horizontal representation of the stack might seem less natural for a computer scientist, it is more convenient to draw and, moreover, it is visually linked to the previous activity with LEGO bricks, where pupils sat in a row.

The software is organized in 'levels' as is common practice in video games. This allows the pupils to observe the execution of the algorithm from view points of increasing depth.

In the first level the goal is to understand the task accomplished by the algorithm, *i.e.,* reverting the input string, and only the relation between input and output is shown. Hence, in the first level, the computer is a "black box" simply receiving the string as input and returning its reversed form.

In the next levels it is possible to conduct experiments through clicking on a button (`EXPLORE`), with the effect of pausing the execution. In order to promote abstraction and generalization by pupils, it is not allowed to pause and observe the behaviour of consecutive function calls. When an experiment ends, it is possible to restart the process and try additional experiments.

The second level shows the process from a global point of view, highlighting the concept of *delegation*, via the *calls* of the recursive function, and the role of *arguments* and *return values* of these calls (see Figure 4). Clicking on `EXPLORE` shows two pieces of information related to the current process state: the fraction of string contained in the tower and the number of letters detached from the input string, which equals the recursion stack size.

The third level shows the operations that are executed during any single call of the recursive function, and makes evident that they are the same for each call, base case excluded (see Figure 5). When one clicks on `EXPLORE`, the small circles become active elements: clicking on one of them has the effect of starting an animation showing the behaviour of the fairy corresponding to that circle, as well as that of the two fairies corresponding to the adjacent circles. All actions performed by such fairies will depend on their position w.r.t. the tower, and on the direction of the tower itself. For instance, if the tower is moving leftwards (Figure 5(b), the fairy sticks the letter she is holding at the basis of the tower, then passes the tower to the fairy on its left.

C. Comparison of previous algorithms

In the third step the two algorithms (string length and reverse) were compared by reviewing their key features and highlighting their commonalities, in a discussion involving the whole class. To start the discussion, the worksheets that guided the second step were reviewed and the answers read aloud and commented by pupils.

D. Guided design of a recursive algorithm which computes powers

In the fourth step the facilitator guided the pupils to design a recursive algorithm that computes 2^{18}, where 18 is the number of pupils in the classroom[5]. The context was rather different from the previous ones, since integer powers, apparently, have nothing in common with strings.

[5]We aimed at the natural, 'tail recursive', algorithm. In [8] there is a nice example which exploits recursion to achieve a faster exponentiation, but it requires an acquaintance with the properties of integer powers which the pupils did not have.

This step aimed at further consolidating the concepts previously encountered. The consolidation was fostered by facing a different context from the first two steps and a new task: the *design* of a recursive algorithm. We point out that the gap between the tasks faced so far and this new one would be too large for the pupils to be able to reach the goal autonomously. In fact, designing an algorithm is much more difficult than analyzing an already implemented one [22]. Furthermore the object of the faced activity—integer powers—was more abstract. Thence this design activity was carried out under the facilitator's guidance.

V. Assessment and evaluation

Assessing the effectiveness of this workshop is not easy. First of all, it lasted only a few hours and was alien from curricular activities, both in the content and in the methodology. Moreover, we did not aim at the development of specific operational skills or competences but had established general or introductory goals. Finally, there are no benchmarks on this topic for this age group.

Hence we decided not to administer profit tests, but rather to found our evaluation on the information gathered by observing the reactions and contributions of pupils during the activities, and by reading their answers to the questions in the worksheets and the reports of the activities they were asked to write a couple of days after the workshop. Such data (since the pupils worked in pairs we collected nine worksheets) are reported in the following: as a whole, they show an acceptable understanding of the concepts, and indicate that at the end of the workshop the participants were convinced that the recursive approach may indeed work.

We also collected the qualitative perception of a teacher who knew the pupils (she teaches them mathematics and science), but was not directly involved in the activities: *"First of all, I noted that pupils were happy to participate in the activities. The activities did engage pupils and gave them an opportunity to look at things from the different point of view of a singular technique which was for sure new to them. I considered particularly effective the combination of an initial kynesthetic step with another one providing the use of computers in order to tackle the* problem-solving *theme."*

A. Reactions and contributions by pupils during the activities

During the discussion that followed the experiments with the LEGO bricks procedure, pupils showed they have understood what had happened during the process and, with the help of suitable questions and stimuli, they were able to notice some key points of recursion as a technique to count the letter of a word. In particular they noticed that: the written instructions were the same for each pupil, even though each of them executed slightly different operations; the task was accomplished by composing the contribution of different actors; each pupil solved a part of the problem and then relied on another mate to complete the task, until a very simple case was reached; such simple case is especially relevant because it is the only one which is resolved completely without involving other mates.

After the experiments with the software, the worksheets were reviewed and the answers discussed with pupils; most

Fig. 4. Level 2 shows the process as a blind delegation.

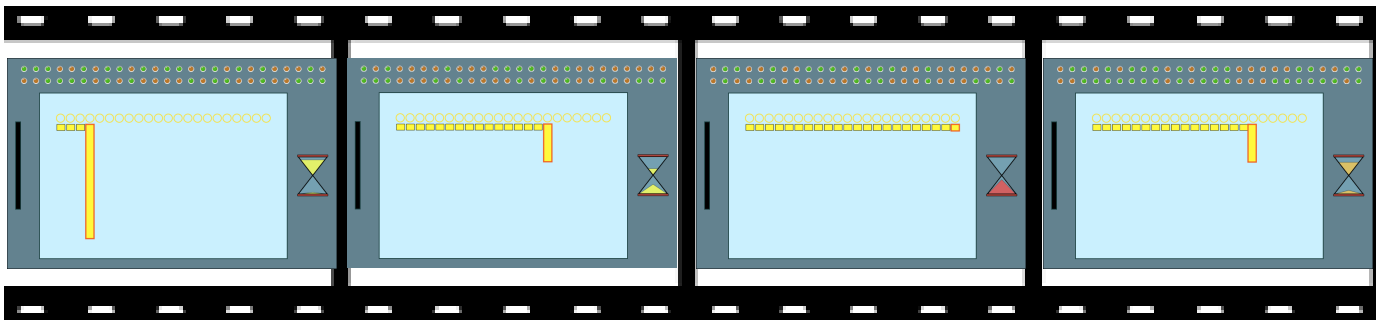
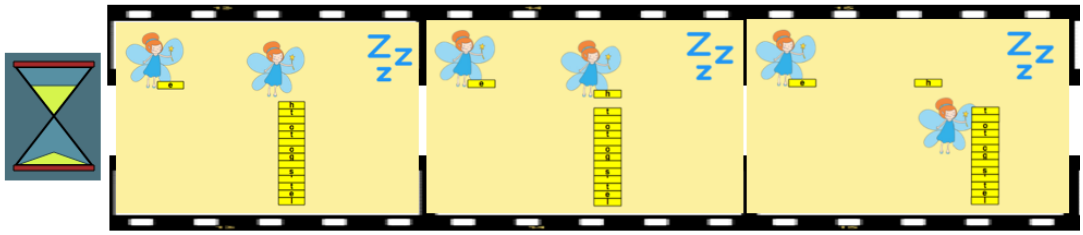
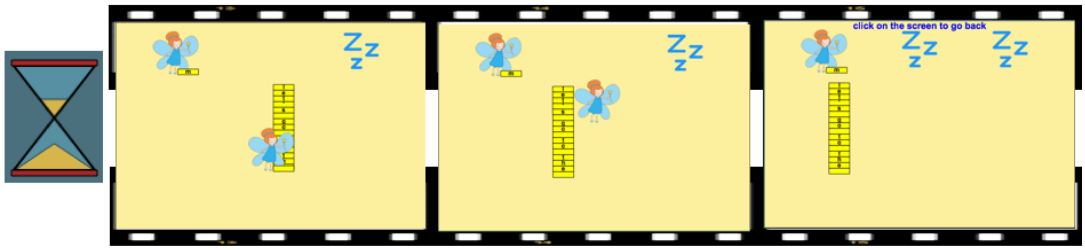


Fig. 5. Level 3 shows the operations executed during any single call of the recursive function: they are the same for each call, base case excluded.



(a) First phase: going from left to right of the screen.



(b) Second phase: going from right to left of the screen.

written answers were correct and the exchange of views among pupils allowed all minor errors or gaps to emerge and to be clarified by the contribution of pupils selves.

During the discussion focusing on the comparison between the two algorithms (string length and reverse), all the basic features of structural recursion emerged, since pupils pointed out (clearly with non-specialistic terminology) that:

- both algorithms operate on a brick tower (*input string*) and pass it around;
- a sequence of actors (the pupils in one case, the fairies in the other case) cooperate; for each letter of the string there is an actor (*each actor executes a recursive function call*);
- each actor (except the last one) performs the same sequence of actions on the tower (*executes the same instructions*);
- each actor (except for the last one) asks another actor to solve the problem on a smaller portion of the tower (*delegation, recursive call on a smaller argument*) and waits for the answer in order to complete his/her work (*return value*);
- all actors do some operations on a tower which could have different sizes (*autosimilarity*);
- the instructions executed by pupils and fairies are relatively simple, but the cooperation between the actors lets them solve a more complex problem (*divide and conquer*);
- if an appropriate number of cooperating actors is available, the algorithm works for any string (*it is valid for each instance of the problem*);
- the last pupil and the last fairy of the sequence receive a tower made up of only one brick and they don't work on it, but just reverse the "direction" of the process (*base case of the recursion*);

In particular, pupils inferred and said that the instructions in the two activities (the ones on the notes handed out in the first activity and the ones executed by the fairies but not visible in the second activity) had to be similar. They also asked us to see the real instructions given to the fairies in the computer, a question which we had not expected. We showed the actual underlying JavaScript code; this request gave us also the opportunity to introduce *self-reference* (a recursive

Fig. 6. Pseudo code for the recursive function executed using LEGO bricks.

```
When a mate on your right asks the power with exponent e,
follow these instructions:
if e is 1
  whisper 2 to your mate.
else
  decrease e by 1;
  ask to the mate on your left to compute the power
   with the decreased exponent;
  wait for his/her answer;
  multiply his/her answer by 2;
  whisper the result to the mate who had asked you.
```

function calls itself until it ends up in the base case), a relevant aspect when implementing a recursive algorithm which was not one of our goals and hence had not been highlighted yet.

In the final step of the activity, after briefly reasoning with the pupils on the computation of powers, the class was able to collectively design a recursive algorithm similar to the ones seen before based on the property $2^n = 2 \times 2^{n-1}$, see Fig. 6.

The pupils then executed the algorithm with a kynesthetic modality, each pupil executing a function call, and observed the general process. During the execution, the role of self-similarity and delegation appeared to be well recognized by everyone, since each pupil needed to engage personally. Pupils were impressed in particular by the fact that the first pupils called upon in the process were actually the ones that had to calculate the last steps of the multiplication, so they had to face the biggest calculation, which highlights the singularity of the recursive approach. Some even noted that the concept of integer powers, that they had found difficult, was now more clear under this new viewpoint.

After the cooperative design and execution of the procedure, pupils were asked to write such a procedure down, which required them to rework on and put into words what they had understood so far. To do so they were allowed to refer to the written instructions they had received in the first activity. Indeed we ascertained that all their written procedure presented a clear recursive structure adapted to the new goal.

B. Analysis of worksheet answers and reports

The worksheet was composed by nine open questions concerning the algorithm which was showed in execution and animated by the tool described in Sect. IV-B. The nine questions asked to identify and interpret certain components of the animation (*e.g.,* "what does the blue number represent?"), to infer properties related to specific moments of the execution (*e.g.,* "how does the tower change when the hourglass is yellow?") or to predict the behavior of the algorithm in a given precise situation (*e.g.,* "look at his picture: which letters are written in the tower, soon after the action of the fairy?"). Such behavior and properties, however, could not be tested directly on the software (namely, the questions referred to inputs having length lower than those accepted by the tool), hence they required to autonomously conceive meaningful experiments in order to generalize the observed behavior and characteristics.

The answers to those questions that asked for a precise prediction proved to be almost always correct; on the contrary, the answers to those questions that asked to verbally describe some features of the algorithm were still basically correct but in some cases they appeared vague or incomplete. For instance some groups wrote that the tower shrinks/grows, some others wrote that it moves rightward/leftward, and only two described both these aspects. In particular, when asked to list the actions performed by fairies they all mentioned verbs like 'take', 'detach', or 'remove' letters, and sometimes 'attach' them; only few reported also 'sleep' and 'wait' and no one mentioned 'pass' and 'receive' the tower. We ascribe these lacks to the young age of pupils, who are not yet able to accurately describe complex phenomena, rather than interpreting them as an actual incomprehension. Indeed, when told of these faults during the subsequent discussion, pupils had no difficulties in understanding and admitting the missing parts.

Concerning reports, we could only perform a qualitative analysis of their content. In fact, their level of detail and precision varies greatly, although in roughly half of them some commonalities arise: specifically, recursion is described as an approach to the solution of different kinds of problems and, to a lesser extent, it is highlighted that it allows to distribute work among executors with the aim of minimizing their effort; moreover, delegation and problem decomposition are mentioned often. It is interesting also to note that some reports assert that at the beginning recursion appears uselessly intricate but then it does work.

VI. Conclusions and further development

Even though in our knowledge there are very few proposals dealing with recursion for our reference age group, we strongly believe in the long-term educational value of such a theme. Also, providing concrete examples of such a fascinating facet of the informatics discipline since the early education stages has a twofold benefit: on the one hand pupils will be empowered in their computational thinking skills, on the other hand our discipline can eventually shine for its more scientific, exciting, and engaging aspects.

Preliminary results show that, relying on self-similarity and delegation, recursive strategies may indeed be within the proximal development zone of pupils. Thus, the activities we designed were successful at least in demystifying the belief that recursion is too an exoteric topic for lower secondary schools pupils.

The proposal could be expanded, for instance to include a classic topic like the binary search. A nice input could be given by the Bebras [23] task depicted in Figure 7. Such task proved to be very hard for pupils of this age but we believe that it could be worked out in our context, provided the fact is pointed out that visitors need not be interviewed in the order they entered the room. Hence, using the inquiry-based approach as above, pupils could be asked to work in groups of two or three to design an efficient recursive strategy to find the thief.

This kind of activities could be structured to compose a learning unit, useful to promote also more operational abilities like:

Fig. 7. A nice starting point to address the binary search.

2016-BE-02 Find the thief

OH NO! The famous Blue Diamond was stolen from the museum today: a thief has swapped it for a cheap imitation with a green color.There were 2000 people who visited the diamond exhibition today. They entered the diamond room one by one. Inspector Bebro must find the thief by interrogating some of these visitors. He has a list of all 2000 visitors in the order they entered the room. He will ask each person the same question: *Did the diamond have the color green or blue when you saw it?* Each person will answer truthfully, except for the thief, who will say that the diamond was already green.

Inspector Bebro is very clever and will use a strategy where the number of people interviewed is as small as possible. Estimate the number of interrogations required to detect the thief

- identify the self-similar structure of a problem even when not evident nor suggested;
- recognize those structures that can be suitably exploited to build recursive solutions;
- establish if a recursive definition is well posed (for instance w.r.t. the base case);
- distinguish among a recursive and an iterative strategy;
- given a problem whose instances have an evident or suggested self-similar structure, be able to write an algorithm solving it recursively;
- design a recursive algorithm for a problem whose structural self-similarity is not highlighted beforehand.

To assess the effectiveness of such a learning unit, one should measure the achievements of pupils w.r.t. knowledge and skills, and an interesting point of view would be relating them to typical misconceptions that occur when recursion is concerned [7].

ACKNOWLEDGMENT

The authors would like to thank Manuel Previtali, the school 'Istituto Comprensivo Ilaria Alpi', and the teacher Martina Palazzolo for their collaboration.

REFERENCES

[1] Violetta Lonati and Mattia Monga and Anna Morpurgo and Mauro Torelli, "What's the fun in informatics? Working to capture children and teachers into the pleasure of computing," in *Informatics in schools: situation, evolution and perspectives (ISSEP 2011)*, ser. Lecture Notes in Computer Science, Kalaš, I. and Mittermeir, R.T., Ed., vol. 7013. Springer-Verlag, 2011, pp. 213–224.

[2] C. Bellettini, V. Lonati, D. Malchiodi, M. Monga, A. Morpurgo, and M. Torelli, "Exploring the processing of formatted texts by a kynesthetic approach," in *Proceedings of the 7th workshop in primary and secondary computing education*, ser. WiPSCE'12. New York, NY, USA: ACM, 2012, pp. 143–144.

[3] C. Bellettini, V. Lonati, D. Malchiodi, M. Monga, A. Morpurgo, M. Torelli, and L. Zecca, "Extracurricular activities for improving the perception of informatics in secondary schools," in *Informatics in schools. teaching and learning perspectives*, ser. Lecture Notes in Computer Science, Y. Gülbahar and E. Karataş, Eds., vol. 8730. Springer International Publishing, 2014, pp. 161–172.

[4] Violetta Lonati and Dario Malchiodi and Mattia Monga and Anna Morpurgo, "Is coding the way to go?" in *ISSEP 2015*, ser. Lecture Notes in Computer Science, Andrej Brodnik and Jan Vahrenhold, Ed., vol. 9378. Switzerland: Springer, 2015, pp. 165--174.

[5] Carlo Bellettini and Violetta Lonati and Dario Malchiodi and Mattia Monga and Anna Morpurgo and Mauro Torelli and Luisa Zecca, "Informatics education in Italian secondary school," *ACM Transactions on Computing Education*, vol. 14, no. 2, pp. 15:1–15:6, 2014.

[6] C. Rinderknecht, "A survey on teaching and learning recursive programming," *Informatics in Education*, vol. 13, no. 1, pp. 87–119, 2014. [Online]. Available: http://www.mii.lt/informatics_in_education/htm/INFE235.htm

[7] D. Dicheva and J. Close, "Mental models of recursion," *Journal of Educational Computing Research*, vol. 14, no. 1, pp. 1–23, 1996. [Online]. Available: http://dx.doi.org/10.2190/AGG9-A5UD-DEK0-80EN

[8] M. M. Sysło and A. B. Kwiatkowska, "Introducing students to recursion: A multi-facet and multi-tool approach," in *ISSEP 2014*, Y. Gülbahar and E. Karataş, Eds. Springer, 2014, pp. 124–137.

[9] L. Vygotsky, *Mind in Society: Development of Higher Psychological Processes*. Cambridge: Harvard University Press, 1978.

[10] B. Haberman and H. Averbuch, "The case of base cases: Why are they so difficult to recognize? student difficulties with recursion," in *Proc. of ITiCSE 2002*. New York, NY, USA: ACM, 2002, pp. 84–88. [Online]. Available: http://doi.acm.org/10.1145/544414.544441

[11] C. Mirolo, "Is iteration really easier to learn than recursion for CS1 students?" in *Proceedings of the Ninth Annual International Conference on International Computing Education Research*, ser. ICER '12. New York, NY, USA: ACM, 2012, pp. 99–104. [Online]. Available: http://doi.acm.org/10.1145/2361276.2361296

[12] S. M. Haynes, "Explaining recursion to the unsophisticated," *SIGCSE Bull.*, vol. 27, no. 3, pp. 3–6, Sep. 1995. [Online]. Available: http://doi.acm.org/10.1145/209849.209850

[13] Code.org, "Code.org," https://code.org.

[14] D. A. Kolb, R. E. Boyatzis, and C. *et al.*. Mainemelis, "Experiential learning theory: Previous research and new directions," *Perspectives on thinking, learning, and cognitive styles*, vol. 1, pp. 227–247, 2001.

[15] C. E. Hmelo-Silver, "Problem-based learning: What and how do students learn?" *Educational Psychology Review*, vol. 16, no. 3, pp. 235–266, 2004.

[16] A. Giordan, "From constructivism to allosteric learning model," http://www.ldes.unige.ch/ang/publi/articles/unesco_AG_96/unesco96.htm, 1996, UNESCO Conference on Science Education 2000+.

[17] Carlo Bellettini and Violetta Lonati and Dario Malchiodi and Mattia Monga and Anna Morpurgo and Mauro Torelli, "What you see is what you have in mind: constructing mental models for formatted text processing," in *Proceedings of ISSEP 2013*, ser. Commentarii informaticae didacticae, no. 6. Universitätsverlag Potsdam, 2013, pp. 139–147.

[18] B. Harvey, *Computer Science Logo Style*, 2nd ed. MIT Press, 1997.

[19] P. D. Palma, "Viewpoint: Why women avoid computer science," *Communications of the ACM*, vol. 44, no. 6, pp. 27–30, 2001.

[20] D. Baranovskiy, "Raphaël," http://dmitrybaranovskiy.github.io/raphael.

[21] H. Kahney, "What do novice programmers know about recursion," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '83. New York, NY, USA: ACM, 1983, pp. 235–239. [Online]. Available: http://doi.acm.org/10.1145/800045.801618

[22] L. W. Anderson, D. R. Krathwohl, P. W. Airasian, K. A. Cruikshank, R. E. Mayer, P. R. Pintrich, J. Raths, and M. C. Wittrock, *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Abridged Edition*, 2nd ed. Pearson, Dec. 2000.

[23] V. Dagienė and S. Sentance, "It's computational thinking! bebras tasks in the curriculum," in *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*. Springer, 2016, pp. 28–39.