Australasian Journal of Educational Technology

2010, 26(3), 309-326

Preschool cookbook of computer programming topics

Leonel Morgado
Universidade de Trás-os-Montes e Alto Douro and GECAD

Maria Cruz
Universidade de Trás-os-Montes e Alto Douro

Ken Kahn
University of Oxford and London Knowledge Lab

A common problem in computer programming use for education in general, not simply as a technical skill, is that children and teachers find themselves constrained by what is possible through limited expertise in computer programming techniques. This is particularly noticeable at the preliterate level, where constructs tend to be limited to extremely simple elements. Having worked for 4 years with over 150 children, aged 3 to 5, we present age-appropriate computer programming activities involving different computer programming techniques, from the basic (computer language syntax) to the advanced (client-server). These may contribute to expand the panorama on viable computer programming techniques available to children and educators, thus broadening the variety of educational activities and projects that can be approached in educational settings using computer programming.

Computer programming by preschoolers and kindergartners

Computer programming is the craft of specifying the behaviour of a computer. It is employed by computer professionals who create applications or games for use by the general public, by professionals in other fields to produce computer tools or calculations to suit their own purposes, by artists to create procedural methods to render visual elements and sound for pictures and movies, and by hobbyists and enthusiasts as a way of self expression. In education, teachers and learners employ programming to explore meanings and rules: since a computer program only works as intended when a programmer conveys concepts and intentions in exact and precise ways, learners create programs to express rules and ideas. To do so, learners need to become aware of shortcomings in the exactness and clarity of their conceptions. Also, learner-made rules can be applied by computers to large numbers of elements or applied a large numbers of times. In observing the outcomes, learners can acquire insights on how complexity can arise from simple rules and concepts, on the impact of small changes to starting conditions, and other abstract concepts linked to mathematics and sciences.

That is the potential, at least (documented since the 1960s, as mentioned ahead). It's limited by the variety and complexity of computer programming techniques that teachers and learners have available to express rules; to craft programs matching the level of complexity of the rules they intend to express. So, to leverage the potential of computer programming for education, one needs to learn about computer programming.

Undoubtedly, while using computer programming to learn other subject matters, be they mathematics, social science, biology, language – or whatever – one also learns about computer programming itself. And yet, at least for adults and adolescents, it's difficult to learn even simple computer programming (e.g. Bonar & Soloway, 1983). A haphazard learning of computer programming, when it is not the focus of either learner or teacher/educator, is often limited in scope. Children and teachers end up using and re-using the same limited set of programming techniques, and going beyond them requires deep personal involvement, in terms of both time and mental effort. As a consequence, teachers and students often end up unable to take intended paths of study using computer programming (Hoyles & Noss, 1999).

Computer programming use for education in general, and not simply as training in a technical skill, has been researched since the 1960s. Research looking for some "magical" automatic impact upon learning from the mere use of computer programming (also known as "Programming as Latin") has not provided consistent results. However, it has shown that when computer programming is employed not by itself but in support of other content, with teacher planning and mediation used (and playing a key role), results are mostly positive across areas as diverse as mathematics, problem solving, higher order thinking, language arts, creativity, and even social-emotional development (Clements, 1999). But there are fundamental differences in goals between teaching programming at a professional level, to college students or adolescents, and simply being able to use it for educational purposes other than programming itself. And the latter is the overall case with children and their teachers, who don't need to control all aspects and details of programming, but rather take advantage of it, and – as much as possible – avoid getting constrained.

To support them, a significant and expanding body of knowledge is available on computer programming use in education by literate children, of formal school age, i.e., aged 6 and up (Clements, 1999). However, for preliterate children, aged 5 or younger, the situation is quite different. Some may argue that children may not be "ready" to program, in particular by considering the well-known Piagetian concept of developmental stages, which preclude "formal" operations (logic reasoning, formulation and testing of hypotheses, abstract thinking) for ages younger than 11. However, these stages have in recent years come under criticism, in light of new research. For instance, recent research has shown that babies are able to employ principles of continuity and cohesion (Baillargeon, 2008). Foremost among this criticism are the stages' focus on "average" children, not accounting for individual differences; the strict demarcation provided by the stages, rather than a progressive, continual evolution; and – foremost for this article – its universal, single progression towards formal thinking and underestimation of the abilities of babies and young children (Papalia, Olds & Feldman, 1999). Bers & Horn (2010) provide research support for computer programming as a developmentally appropriate practice in preschools.

The body of existing research on computer programming with children younger than 6 is fragmentary and insufficient for preschool or kindergarten teachers to efficiently plan and develop computer programming activities. But it does tell us that very young children can indeed program – not just store a sequence of actions to be repeated later, but the actual ability to create that sequence; to include conditions for different cases and generalise that sequence to render it usable to other cases, not just exact repetitions of the original actions; to recognise flaws in the sequence; to revise it, etc. The first

obstacle towards this was the text-based nature of original programming systems. But this obstacle has long been tackled: using tangible programming systems (e.g. Perlman, 1974; Wyeth & Wyeth, 2001); reducing commands to single keystrokes (e.g. Goldenberg, 1974); using keyboard overlays and adaptations (e.g., Correia, 1990); employing icon-based or robotics-based systems (e.g. Bearden & Martin, 1998; João-Monteiro, Cristóvão-Morgado, Bulas-Cruz & Morgado, 2003; Janka, 2008); and selecting pre-typed words that can be read aloud by the system (Kindborg & Sökjer, 2007).

Research efforts with preschool-aged children typically don't focus on the actual programming techniques being used. They seem to be planned as if any programming at all would suffice, regardless of its nature. Other research efforts focus exclusively on the techniques, not on any context for them, posing technocentric questions: researching the impact of programming on children as if it were an independent variable, rather than looking at how it was used with whom, specifically, it was being used, their conclusions understandably contradictory (e.g. Degelman, Free, Scarlato, Blackburn & Golden, 1986). Other research efforts, still, are based on extremely limited interactions between preschoolers and the researchers developing the systems (e.g. Zuckerman & Resnick, 2003), or are focused on older children, approaching preschoolers as little more than an extra trial (e.g. Clements & Battista, 2001; Horn et al., 2009). Even when research efforts describe the programming that took place, they usually do so without looking into the educational processes that are taking place or the educational settings, thus providing results useful for the overall understanding of potential outcomes from programming activities, but not much help regarding how those activities should be planned and conducted (e.g. Hopper & Lawler, 1997; Erwin, Cyr & Rogers, 1999; Raffle, 2004; Smith, 2008). Only a handful of studies provide data on settings and context (e.g. Wyeth & Wyeth, 2001; Bers, Ponte, Juelich, Viera & Schenker, 2002; Wyeth & Purchase, 2003; Wyeth & Wyeth, 2008).

The need for a cookbook of children's programming topics

In our view, the ideal situation for preschool/kindergarten contexts would be to have a computer programming consultant available to teachers/learners. A consultant that is focused on educational needs and realities, a partner of teachers and students. Should a teacher/learner come across a situation where his/her programming skills were insufficient, the consultant could suggest an adequate new programming technique or approach. This way, even the learning of programming would occur with a specific goal, embedded in the overall educational goals, not just for its own sake. This is similar to the well-known approach for artistic techniques used in preschools of the Italian city of Reggio Emilia, where art educators (i.e. the equivalent to our proposed "computer consultants"), not just teachers, are involved in the educational activities (New, 2000). For most preschool/kindergarten education settings, this ideal situation regarding programming is not an available option, either now or in the near future. Therefore, our aim here is to provide some help for teachers and learners who are using computer programming on their own, and for computer scientists aiming to develop computer programming systems and research geared towards early childhood education. It is a list of various programming techniques, their translation in terms of *ToonTalk*, an animated, action-based programming language (*ToonTalk*, n.d.), and ideas for applying those techniques in educational activities – i.e., uses beyond the learning of programming.

We've called it "cookbook", because that's how we see its usefulness: a teacher should not feel the obligation to try all these techniques. Rather, our intention is that this resource allows teachers to try out a new computer programming concept in the context of everyday educational activities. Most people who read cookbooks don't feel pressed to cook all the recipes in them, or to follow them rigidly. One browses cookbooks in search of dishes that one may "feel like" trying at a given occasion, either because they appeal to one's tastes or suit the ingredients (or time) available, and often adapts the recipe to the time and ingredients on hand, or as a basis for personal creations. By trying a new "programming dish" from time to time, or at least being aware of the number of "dishes" that he/she still didn't "cook", a teacher/learner may use this section to advance programming knowledge with hands on uses beyond programming itself, "making learning worth while for use now and not only for banking to use later" (Papert, 1999, p. xvi).

The *ToonTalk* animated, action-based programming language

Preschoolers' limited ability to conceptualise rules and abstractions may be seen, from a broader perspective, as the main issue at stake, together with the absence of basic reading and writing skills. Further, young children's fine motor skills are often inadequate for an accurate control of a computer mouse. This renders inadequate several techniques used in many systems (such as dragging small handles of objects on the screen). Amongst existing programming systems for children, we chose *ToonTalk*, which is an animated, action-based programming system and programming language (Morgado & Kahn, 2008). In *ToonTalk*, the programmer is immersed in a virtual world (a virtual city), and programs by performing live actions with an avatar. That is, the programmer programs by moving around, picking up objects, and using virtual tools such as a vacuum cleaner or a bike pump (Figure 1). For instance, dropping a number on top of another adds them, and doing the same with images combines them. Sequences of actions can be "taught" to robots by demonstration, and the specific case demonstrated to a robot is stored as "starting condition": more precisely, as pictures in the robot's "thought balloon". These robots can then be generalised, so they become real generic programs, not just collections of specific actions, by using an animated manipulation method: "erasing" the surface of objects or "vacuuming" them from the robot's thought balloon (Kahn, 2001). This allows young children to perform complex programming actions by contextualised and concrete manipulation of objects, instead of relying on reading or in the abstract interpretation of an alphabet of action-depicting symbols. In other programming systems for children, the programmer is not part of the environment; rather all takes place in more traditional interfaces (e.g., Smith, Cypher & Tesler, 2001; Steinmetz, 2001).

ToonTalk was originally created over 10 years ago. Since its creation, it evolved through numerous refinements, additions, and changes to the language and development environment, inspired not only from user feedback but also from several educational research projects (e.g., Noss, Hoyles, Gurtner, Adamson & Lowe, 2002), the current technical status of the language having been published recently (Morgado & Kahn, 2008).

Educational methodologies have also been developed (e.g., Mor, Noss, Hoyles, Kahn & Simpson, 2006), the most relevant to this article being the immersion of programming in a virtual world to provide a simple way to create rich context settings, within which programming activities can be conducted (Morgado, 2008). Rich context

represents a constant support for young children's plans, therefore mitigating the necessity to memorise a specific sequence of strict plan – a feature that is quite relevant at early ages.

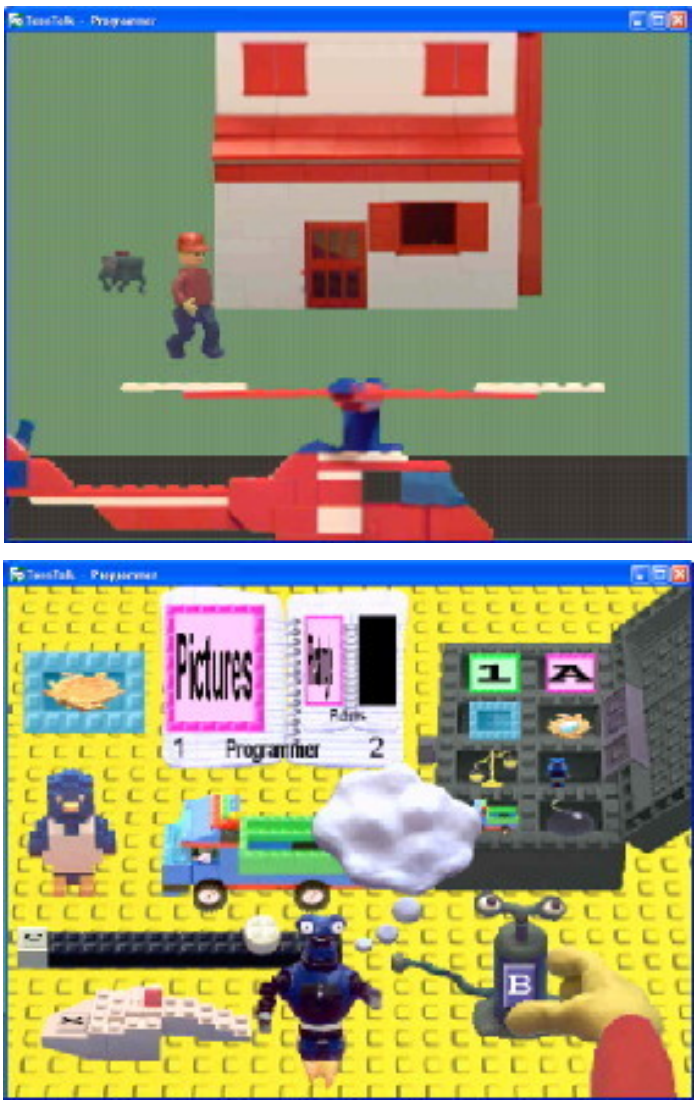


Figure 1: *ToonTalk* programming environment

Field research background

The activities proposed in this cookbook were inspired by a 4-year research project employing the action research methodology, which involved over 150 children, aged 3 to 5 (Morgado, 2005, pp. 364-385). Session reports are publicly available (ibid.), and data in other media are available for researchers upon request. Some videos are online, as mentioned further ahead.

Initially, exploratory research activities were conducted for two months with 7 children (two at a time, inside the preschool room), to establish the feasibility of several issues: the programming tool, the duration and frequency of programming sessions, the intervention approach, and children's ability to use *ToonTalk's* generalisation method (Morgado, Cruz, & Kahn, 2003). A first action research iteration was then conducted with 21 children over 5 months, and data from this iteration (session reports, screenshots, recordings of screen activity) was then analysed, reflected upon, and informed a second action research iteration, with 6 children, recording the same types of data over 4 months. The exploratory activities and the action research settings differing variables were the location (inside/outside preschool room) and the number of children per session (one pair of children at a time/a group of 3 pairs at a time).

By reflecting upon the data and knowledge from this action research process, in the following iteraction a different variable was explored: the sessions became mediated, by using identical settings but having the researchers instruct and supervise teachers. Besides the session reports and other data recorded by teachers, weekly meetings were held between teachers and researchers, and reports made. This iteration involved 6 teachers and 99 children in various preschools, for 6 months.

Given that the number of sessions in this research with children aged 3 had been so far limited, the following iteration resumed the non-mediated context and settings, but focused exclusively on 3-year olds, involving 4 children for 5 months.

The data and knowledge from these action research iterations fed into another iteration with mediation, this time further removing contact between researchers and teachers. It took place over two settings: in one, a researcher instructed two early childhood education teacher trainees on activities and methods, and explained to them the requirements for making records and collecting information. The activities ensued with 21 children, with no more researcher-teacher contact taking place until information was collected from the trainees, at the end of that academic semester. In the other setting, a single five year old child in the USA attended sessions with an adult from his family for two months, with no contact between researchers and that adult other than a weekly email exchange before each session, to collect data and provide orientation for the following session.

Overall, children enjoyed the programming activities, with many often asking their teachers and parents to use *ToonTalk* beyond the scheduled sessions, even months after their participation. But this research did not focus on establishing the abilities and learning of children beyond programming or because of it. Rather, since many research efforts had demonstrated previously the feasibility of the concept and its potential, our focus was towards determining how to support its use in actual school environments, beyond research settings. Thus, the main contributions of this research effort were an identification of hurdles preschoolers face while developing programming activities, and an organisation of settings and methods for those activities under the form of a framework (Morgado, 2005).

Cookbook recipes

Here we present a representative sample of cookbook recipes. Others are mentioned at the end of this document. The descriptions can be difficult to follow because they

describe highly dynamic animations with static words and pictures on a printed page. We are placing some sample videos online so interested readers can watch the animation (and children interaction), at http://home.utad.pt/~leonelm/cookbook/

Recipe 1: Syntax and semantics

Concept and preschool use
Programming languages, as all others, are not a random assortment of symbols. They must be used in proper form, within specific rules of grammar and usage – syntax. For a computer program, it's the expressions and associated evaluation rules that compose its code. For a child to construct a program, he/she must be able to employ the syntax of a computer language. For instance, in *ToonTalk*, an array is created by joining two boxes, dropping one on either the left or right side of another; any other combination will result in dropping a box inside or alongside the other. That is, if a child does not understand the existence of a syntactical difference (from the computer's perspective) between presenting two boxes nearby and two boxes joined together, this lack of understanding will prevent his/her abiliy to express concepts to the computer. In computing terms, knowing that syntax exists is the realisation that there are strict ways to express concepts to the computer.

Preschool education connection
Understanding syntax and semantics at this level can be useful, in general educational terms, because it provides an opportunity for understanding a rule set in a context-rich environment, and a window into the notion of how different people can interpret the same thing differently.

Activity guidelines
In a virtual animated world such as the *ToonTalk* programming environment, there would seem to be little difference between mastering the environment itself (user interface) and mastering the language's syntax, their borderline seemingly tenuous. But even a little experience shows that it is no so. Seemingly "strange" limitations are revealed while using *ToonTalk*: joining boxes can only be done by left or right sides, not by the top or bottom; robots only accept boxes; etc. Seemingly strange events occur: if one drops a picture on top of another, the dropped picture is clipped at the borders of the bottom one. Why shouldn't they both be visible on the floor? And while flipping a notebook, when does one reach its end? (Children sometimes flip pages endlessly, trying to find "the end" of *ToonTalk's* notebooks.) The reason these distinctions exist is the same: syntax. They came to be in order to limit potential programming problems and allow a richer expression of complex ideas. But they pose a challenge for educational use with young children: most of them have apparently no reason to be, from a child's perspective. These "rules of the game" can be hard to grasp in purely "that's how it is" form.

Educational activities should make an effort to bring metaphoric, everyday logic to these rules. For instance, stating that "robots like to tidy up everything, keeping everything nicely stored in boxes", provides a reason for them to accept only boxes. That boxes piled up high can fall and cause injury may be a reason for not allowing them to be linked on the top or bottom; and a picture dropped on another gets clipped at the edge of the bottom picture, because that's where the paper sheet ends (*ToonTalk* wouldn't want the floor all cluttered with paper snippets). Syntax, however, doesn't end there. It's intermixed with semantics, dictating more subtle results. For instance, when teaching a robot, one must learn that two onlookers of the same box may not see

the same thing: in Figure 2, under the *ToonTalk* syntax and semantics, one is teaching the robot to pick up the fourth element, not the last one nor the one containing the value "2". For the child, this provides a glimpse of another mind ("the robot thought you meant 'fourth' ").

Figure 2: The robot is picking "the fourth element", not "the 2" nor "the last"

Recipe 2: Compound procedures

Computer programming concept
Combining of operations into a more complex, compound operation, which can be named and then referred to as a unit. The internal operations can be performed on static arguments, or on arguments provided to the procedure as a unit (parameters).

Preschool education connection
Beyond the concept of acting upon an object ("do to a thing"), using procedures is associated with the abstract notion of an action as a thing – a "thing to do". And that "thing to do" can be a collection of smaller "things to do", the whole set having a name. Being able to envision a set of actions as an identifiable procedure is connected to a skill that helps to understand and solve problems: problem decomposition.

Activity guidelines
Children can define and use procedures, without a computer, through simple games or other activities where the procedures have names such as "when I clap, we all touch our ears and touch our hips." However, beyond simple defining or executing of such procedures, a further step is achieved by starting with a problem, goal, or action and proceeding by dividing it into smaller problems and actions, identifying them as isolated pieces, by naming, branding, or some other identification method, such as "the left one", "the one with birds", etc.

For instance, in *ToonTalk*, it's easy to build new houses (the child loads a truck with a robot and a box, and the truck goes away and builds the new house with that robot inside, working on the box). But if two children build houses together, it's hard to tell which house was built by whom. But children can also easily put their name on the roof of a house (each child picks up a "roof sensor", writes his/her name with letter blocks and drops the name on the roof sensor). By teaching a robot these activities, and sending that robot in the truck to build a new house, the new house will have the child's name (Figure 3). This robot can be identified in various ways, such as "that one", "the one on page 20", "the one called M", "the one near the door", etc. This usage of a robot to put one's name on the roof serves a purpose ("whose house is that?") but also requires understanding that actions taught to that robot will take place as a block, and one can refer to them as such.

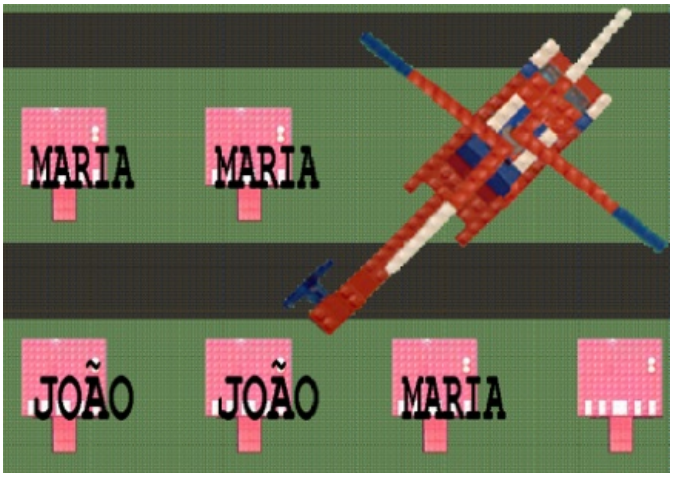


Figure 3: In this city, children can tell which houses are "theirs"

Recipe 3: Parameter passing

Computer programming concept
The operations in compound procedures can be applied to specific static arguments. However, general arguments can be made available for all the operations within a procedure; these are called the procedure's "parameters". When elements/values are assigned as arguments to a procedure, that assignment is called "parameter passing", as if parameters were handed over to the procedure's operations.

Preschool education connection
The distinction between "thing to do", and "do to a thing" is completed with the realisation that the same "thing to do" can be done to different things. For instance, "turn left" can be applied to the top of a jar or to a door knob, with different consequences. This comprises two cognitive steps: deciding what is needed to perform a procedure, i.e., what will constitute the object(s) of the procedure; and realising that such object(s) can be different from what was originally planned (generalisation).

Activity guidelines
In programming (but also in non-programming activities), rather than define a procedure by focusing the activity just on "what to do", one can also focus it on "what is needed". As an example, one can consider the case mentioned in the previous recipe, where a robot would have to be taught how to put a name on the roof of the house. This could involve teaching a robot to pick up text pads, write "João", go to the notebook, pick up the roof sensor, and drop the name "João" on the notebook. But instead, one could conceive a robot that would place on the roof any name it was given. There is a large cognitive leap from teaching a robot how to do a specific thing (write "João" and put it on the roof) to teaching it about "receiving" parameters. That leap can be eased by presenting the activity in a different manner, inquiring: "what do you want to do?"; "– to put my name on the roof"; "what do you need, to do that?"; "– I need my name and the roof". This approach provides a motive for the existence of parameters even in the simplest cases (Figure 4). That is, instead of teaching a robot how to write "João", by teaching it to use a name it receives, we open the door to generalisation.

Generalisation, in *ToonTalk*, can then be achieved by removing detail about the parameters from the robot's thought bubble, using a vacuum cleaner tool. The experience from working with preschool/kindergarten children supports the feasibility of presenting this technique by producing several similar robots (for instance, one to write "João", another to write "Maria", another to write "Miguel", etc.) and present generalisation to children as a matter of making the robots less "picky" (Morgado, Cruz & Kahn, 2003).

Figure 4: Assembling "what is needed" before teaching a robot

Recipe 4: Parallel/concurrent execution

Computer programming concept
Several programs or processes can be executed seemingly at the same time.

Preschool education connection
A problem or goal doesn't have to be analysed as a sequence of steps. Rather, different parts of a problem can be considered independently, just like several children can cooperate, working concurrently towards a common goal.

Activity guidelines
Activities for children can consider the existence of multiple parallel events or actions in view. For instance, while a child conducts his/her activities, robots programmed by other children can be visible running concurrently. Or various automated objects can be employed at the same time, rather than planning for only one to work at a time ("While Julian's robot is puting something on the wall, yours can be puting something on the roof"). And, although *ToonTalk* doesn't currently support it, in a multi-user programming environment the actions of other programmers can originate changes as one child navigates through his/her programming environment. Another option is to consider that the parallel events or actions are taking place away from view. For instance, *ToonTalk* houses can be customised as farm buildings, such as a stable, a pigsty, etc. Various children can tend the animals in different houses; collect fruits, harvest fields, etc.

In a multi-user environment, the changes would be concurrent; in the current single-user *ToonTalk* environment, changes take place sequentially. However, the amount of consecutive time that preschool aged children dedicate to computer activities is fairly limited. As a consequence, every time a child plays in this "shared farm" it seems as if various events are taking place concurrently. A particular event that occurred with a 3.5 year-old girl, during our field work, helps shed some light on these statements. She played with several automated images, whose animations could be turned on and off, using the space bar and the "." key (Figure 5 – the top right and bottom left pictures are turned off, the others are in motion). One week later, she resumed her *ToonTalk* play, from the same point. After *ToonTalk* launched, displaying some of those images working as she left them, she immediately exclaimed in surprise: "they're *still* working!?!"

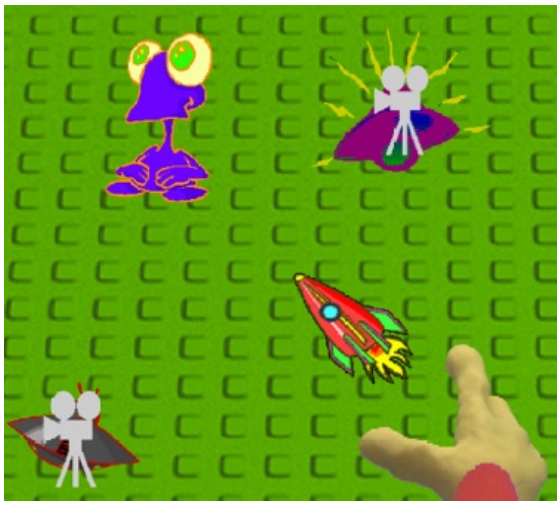


Figure 5: Assorted animating and stopped pictures

Recipe 5: Message passing / communication channels

Computer programming concept
Parallel processes can activate one another and exchange information. One method for this is the exchange of messages. Messages can also be passed between one process and several others, in what is called a multicast, or 1-to-N communication (if a process receives messages from several processes in the same queue, it is called a concentration, or N-to-1 communication). In *ToonTalk*, message communication is achieved by using birds, which act in a way similar to carrier pigeons. A bird can be used to carry an object to its nest, but nests can be joined together, so that several birds share the same nest (N-to-1). Nests can also be copied, so that a bird delivers several copies of an object at the same time, one in each copy of the nest (1-to-N).

Preschool education connection
Exchanges of information are common in childhood activities, involving mathematics or social relationships. This can be not only speech, but also the transportation of objects between locations.

Activity guidelines
All *ToonTalk* activities using birds to carry things involve message passing. At the simplest level, this can be something as simple as offering gifts: two children playing in the same *ToonTalk* session, each with his/her own bird and nest. By exchanging nests, the children can use the birds to send "gifts" to each other. This can initially occur in plain sight (all birds and nests in view), but made more complex, by having a different house for each child. This technique could be used in various ways: a postman that can use the birds to deliver letters and parcels; a baker that can use birds to deliver bread and cakes, as well as receive orders. If the nests are in robots' boxes, the arrival of an object carried by a bird can allow robots to perform as they were taught. This could be used to create reactions to the arrival of messages (such as playing sounds, placing images on the wall, etc.), but also to respond. If the receiving robots have a bird, and the sender has another bird, the robot can "reply" to the message (Figure 6). This has been used, for instance, to make two robots "toss a ball around", but also by a child as young as 3 to "play ball" with a robot he programmed that would send it back (a video is at http://home.utad.pt/~leonelm/cookbook/).

Figure 6: Sending a ball for the robot to return it

Recipe 6: Input guards

Computer programming concept
Input guards provide a way to support synchronised communication between processes. A command with an input guard is executed only if and when the source of data for input is ready to provide its output. In *ToonTalk*, input guards can be seen as nests in the box given to a robot. The robot simply waits until something arrives on a nest, and only then will it compare the constraints with the contents (see if the guard holds) and proceed with execution.

Preschool education connection
Using an input guard implies that children need to define two procedures (robots), and link them together (using birds). To do so, they must be able to keep in mind the actual actions of both robots, and devise a plan for flow of information: "this one will give this to the bird, which will take it to that one, which will do X." In other words, children must go beyond thinking about communication, and start thinking about the necessity of coordinating communications. This can be used in activities to help develop the concept of conservationism (objects that are no longer in sight still exist and operate), since one of the robots can be working in a house (or in another corner of the current house), out of sight.

Activity guidelines
The simplest level occurs when a robot is simply waiting for something to arrive in its nest, and the sender is the child herself (i.e., the child, not another robot, gives the missing object to the bird). This level is represented in examples such as those mentioned in the recipe on communication (Recipe 5). Other simple examples, not requiring the robot to communicate back, are: (a) a robot that vacuums the contents of its box; if one places a nest in that box, from then on, the robot only works when something reaches the nest; (b) an "interior decorator" robot (also used with a nest in its box), which places on the room wall any picture it gets.

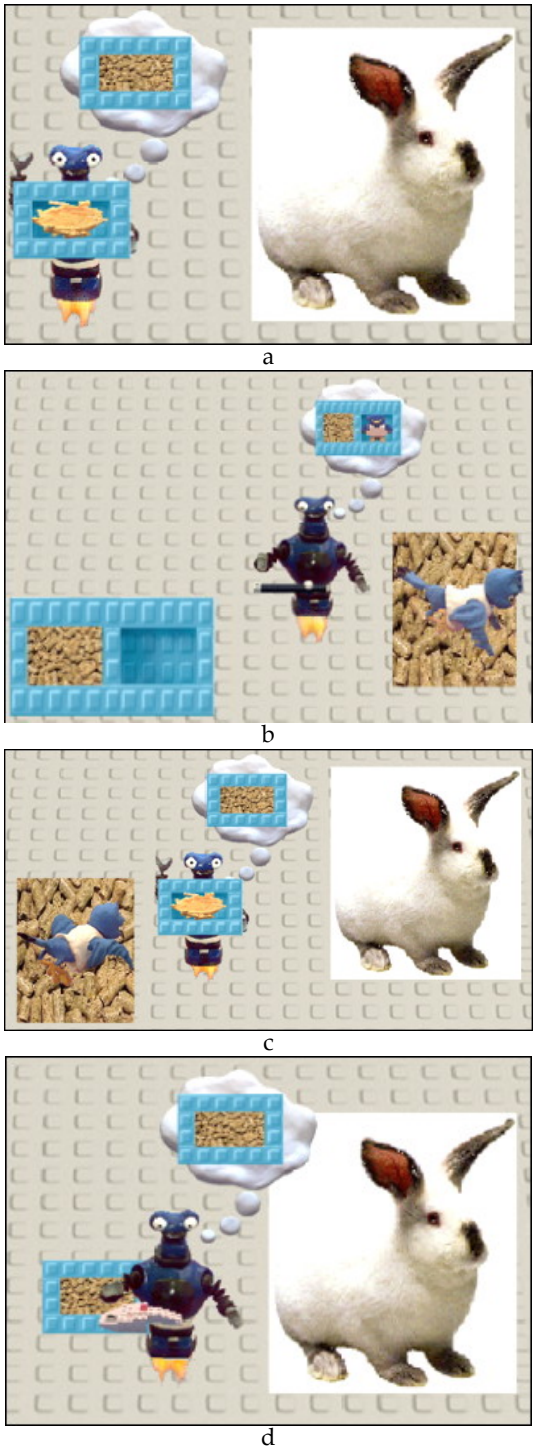
At a more complex level, a robot sends an object to another, which then acts upon the thing received: for instance, a robot sending a ball to another robot, which then replies by sending the ball back. This example, once working, could be changed by moving one robot away from the visible area, or even to another house. And yet, the two robots would still be able to "play ball". But there is no need for a robot to respond, it could instead react: for instance, a robot working in a warehouse could be dispatching Christmas decorations for robots in other houses; those robots, upon reception, could put them on the roofs. By travelling around in *ToonTalk's* helicopter, the child could see which houses had robots working in them that had already received the lights, and which houses either had no robot in to put the lights on the roof or had not received lights.

Conversely, a robot may be at work, dispatching feed to rabbits, being bred in another house (decorated as a rabbit hutch). That feed would be accumulating, even if the nest where it is arriving is placed behind the picture of a rabbit. But a robot could be taught how to vacuum that feed, as if the rabbit had "eaten it" (see Figure 7). Better still, that robot could do something after vacuuming the feed, such as playing a munching sound, or making the rabbit fatter!

Recipe 18: Clients and servers

Computer programming concept
An important concurrent programming style involves the use of client-server relationships between processes. A server process renders a service to one or more client processes, through message exchanges (in *ToonTalk*, using birds, as mentioned). Birds can be fixed at the time of programming (i.e., a server always responds to the same clients) or passed along from the client to the server at program runtime, for dynamic client-server interaction.

Preschool education connection
When modeling real world interactions with a computer program, one finds multiple opportunities for employing the client-server model (shopping and parcel deliveries, for instance). A child programming a client process to make requests does not require knowledge of how they will be answered. She just needs to worry about what are the answers that may arrive (and in fact, whether they do at all or not). This requires the child to reason over the expected behavior of non-local, non-present entities (since the server process usually operates out of sight). Also, there is ample space for activities related to memory use. For instance, what if an answer does not arrive? The child must remember that a request was done earlier, and that an answer is missing. What if it does arrive? She needs to focus and associate the answer with the request made earlier, and be able to check if the answer matches what was intended in the request (besides memory, this is also an opportunity for using record keeping).



a

b

c

d

Figure 7: Feeding a rabbit (sequence: a, b, c, d)

Activity guidelines
A crucial aspect of such activities is how to identify the client making a request, and the server to whom the request is being made. In preschool settings, this can be made with text (capitalised names), but the most versatile solutions would be the use of pictures and/or sounds. For instance, Figure 8 displays several "requests", in the form of *ToonTalk* boxes. In the rightmost hole of each, a picture identifies the "requester", represented by his/her "hat" (nurse, nurse, baker, and postman, respectively). These could also be sounds for the child to play, achieving the same effect. Another aspect is how to identify the request. Again, the use of pictures and sounds provides more flexibility, rather than depending solely on text or numbers (yet, these can also be used).

For instance, in the same figure, the top request comes from the nurse, and it consists of a parcel (first hole) and a postage stamp (second hole). The text in the third hole simply restates this in written form. Numbers could be used to represent how many boxes and stamps are being requested, so that the stamp picture could stand for as many stamps as desired. A child can execute "server" actions, responding to requests, or program a robot to respond, or to produce the answer. The answers can be delivered to the requesters manually, but the requests can also include a reply bird (there can be an extra hole in the request box, for this reply bird, but it can also be behind one of the pictures). Using such "reply birds", server robots or children can answer requests from clients not known beforehand.

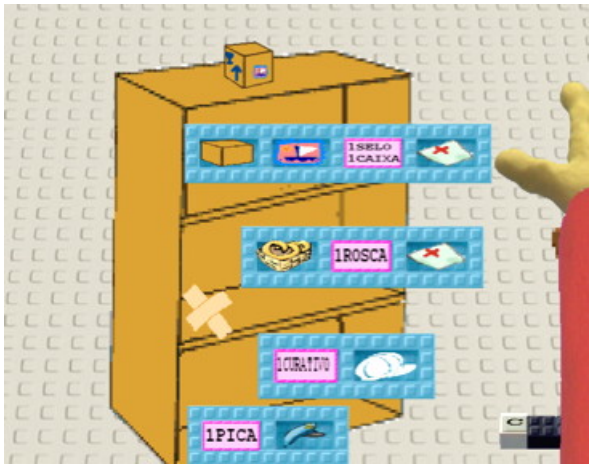


Figure 8: Sorting requests

More recipes

Besides these sample recipes, many others can be developed. The following terms may sound strange for educators, who we hope may wonder how they "taste" like (now that they have sampled some recipes in this paper). It's as if they are new dishes from a cook they enjoyed previously. So, speaking for a moment in terms used by computer scientists, we have also considered (and created) recipes for Computability; Programming environment; Declarations, expressions and statements; Conditional expressions; Type checking; Higher-order procedures; Parallel/concurrent statements; Speed independence; Synchronous/asynchronous communication; Recursion; and

Guards with alternative commands (Morgado, 2005). But above all we hope that people may develop further activities. People wishing to help children use computers in a creative manner. People using computers in support of children's global development and thinking. And we hope those activities become more detailed, richer, funnier, and more imaginative, as children and teachers rightly deserve.

References

Baillargeon, R. (2008). Innate ideas revisited: For a principle of persistence in infants' physical reasoning. *Perspectives on Psychological Science*, 3, 2-13.

Bearden, D. & Martin, K. (1998). My make believe castle – an epic adventure in problem solving. *Learning and Leading with Technology*, 25(5), 21-25.

Bers, M., Ponte, I., Juelich, C., Viera, A. & Schenker, J. (2002). Teachers as designers: Integrating robotics in early childhood education. *Information Technology in Childhood Education Annual*, 2002(1), 123-145.

Bers, M. U. & Horn, M. S. (in press, 2010). Tangible programming in early childhood: Revisiting developmental assumptions through new technologies. In I. R. Berson & M. J. Berson (Eds), *High-tech tots: Childhood in a digital world*. Greenwich, CT: Information Age Publishing. [verified 7 Apr 2010; 7.7 MB] http://ase.tufts.edu/devtech/publications/Bers-Horn_May1809.pdf

Clements, D. H. & Battista, M. T. (2001). *Logo and geometry*. Reston, VA, USA: National Council of Teachers of Mathematics.

Clements, D. H. (1999). The future of educational computing research: The case of computer programming. *Information Technology in Childhood Education Annual*, 1999-1, 147-179.

Correia, R. (1990). A Informática ao Serviço da Aprendizagem. *BICA – Boletim informativo do centro de recursos – Interactividade, Comunicação, Aprendizagem*, 90/91 (1), 11-15.

Degelman, D., Free, J., Scarlato, M., Blackburn, J. & Golden, T. (1986). Concept learning in preschool children: Effects of a short-term LOGO experience. *Journal of Educational Computing Research*, 2(2), 199-205.

Erwin, B., Cyr, M. & Rogers, C. (1999). *LEGO Engineer* and *RoboLab*: Teaching engineering with *LabVIEW* from kindergarten to graduate school. *International Journal of Engineering Education*, 16(3). [verified 3 May 2010] http://www.ijee.dit.ie/articles/Vol16-3/ijee1121.pdf

Goldenberg, P. (1974). FASTR – A simple turtle runner. *Logo Working Paper,* 30, MIT AI Lab. Cambridge, MA, USA: Massachussets Institute of Technology.

Hopper, M. E. & Lawler, R. W. (1997). A progress report for the Headstart-Apple Logo Project. In *Learning with Computers* (pp. 36-40), Bristol, UK: Intellect Books.

Horn, M., Solovey, E., Crouser, R. & Jacob, R. (2009). Comparing the use of tangible and graphical programming languages for informal science education. In *Proceedings of the 27th international Conference on Human Factors in Computing Systems* (CHI '09). Boston, MA, 4-9 April 2009, pp. 975-984. New York, NY: ACM.

Hoyles, C. & Noss, R. (1999). Playing with (and without) words. *Proceedings of the Seventh European Logo Conference Eurologo '99*, Sofia, Bulgaria.

Janka, P. (2008). Using a programmable toy at preschool age: Why and how? In S. Carpin, I. Noda, E. Pagello, M. Reggiani & O. Stryk (Eds.), *Simulation, Modeling, and Programming for Autonomous Robots. First International Conference, SIMPAR 2008 Venice, Italy, 3-7 November, 2008. Proceedings*, pp.112-121. Berlin, Germany: Springer. [verified 3 May 2010] https://www.di.unito.it/~barbara/WorkVenezia3nov08/TeachingWithRobotics/pekarova.pdf

João-Monteiro, M., Cristóvão-Morgado, R., Bulas-Cruz, M. & Morgado, L. (2003). A robot in kindergarten. *Eurologo'2003 Proceedings – Re-inventing technology on education*, Coimbra, Portugal. [verified 3 May 2010] http://home.utad.pt/~leonelm/papers/RobotinKindergarten/RobotinKindergarten.html

Kahn, K. (2001). Generalizing by removing detail: How any program can be created by working with examples. In *Your wish is my command: Programming by example*, San Francisco, CA: Morgan Kaufmann Publishers.

Kindborg, M. & Sökjer, P. (2007). How preschool children used a behaviour-based programming tool. In *Proceedings of the 6th international Conference on interaction Design and Children (Aalborg, Denmark, 6-8 June 2007). IDC '07*, pp. 149-152, New York, NY: ACM.

Mor, Y., Noss, R., Hoyles, C., Kahn, K. & Simpson, G. (2006). Designing to see and share structure in number sequences. *International Journal for Technology in Mathematics Education*, 13(2), 65-78.

Morgado, L. (2005). *Framework for computer programming in preschool and kindergarten*, PhD Thesis, Vila Real, Portugal: University of Trás-os-Montes e Alto Douro.

Morgado, L. (2008). TEL practices in preschool and kindergarten education: Integrating computer use and computer programming in off-computer activities. In M. D. Lytras, D. Gasevic & P. Ordonez de Pablos (Eds), *Technology enhanced learning: Best practices*, Hershey, PA: IGI Publishing.

Morgado, L., Cruz, M. & Kahn, K. (2003). Taking programming into kindergartens: Exploratory research activities using *ToonTalk*. In *Eurologo'2003 Proceedings – Re-inventing technology on education*, Lisbon, Portugal, pp. 178-189. [verified 3 May 2010] http://edutice.archives-ouvertes.fr/docs/00/19/01/34/PDF/Morgado_2003c.pdf

Morgado, L. & Kahn, K. (2008). Towards a specification of the *ToonTalk* language. *Journal of Visual Languages & Computing*, 19(5), 574-597.

New, R. S. (2000). Reggio Emilia: Catalyst for change and conversation. *ERIC Digest* EDO-PS-00-15. [verified 3 May 2010] http://www.ericdigests.org/2001-3/reggio.htm

Noss, R., Hoyles, C., Gurtner, J. L., Adamson, R. & Lowe, S. (2002). Face-to-face and online collaboration: Appreciating rules and adding complexity. *International Journal of Continuing Engineering Education and Lifelong Learning*, 12(5/6), 521-540. [verified 3 May 2010] http://www.lkl.ac.uk/rnoss/papers/face-to-faceIJCEELL.pdf

Papalia, D. E., Olds, S. W. & Feldman, R. D. (1999). *A child's world: Infancy through adolescence*, 8th edition. New York: McGraw-Hill.

Papert, S. (1999). Introduction: What is Logo? And who needs it? In *Logo philosophy and implementation*. Highgate Springs, Vermont: LCSI.

Perlman, R. (1974). TORTIS – Toddler's own recursive turtle interpreter system. *MIT AI Memo* 311, *Logo Memo* 9. Cambridge, MA, USA: Massachusetts Institute of Technology.

Playground Project (2001). *3rd Annual Report – September 2001*. Report to the European Union. London, UK: Institute of Education.

Raffle, H. S. (2004). *Topobo: A 3-D constructive assembly system with kinetic memory* (Master's dissertation). Cambridge, MA, USA: Massachusetts Institute of Technology. [verified 3 May 2010] http://dspace.mit.edu/handle/1721.1/26920

Smith, D. C., Cypher, A. & Tesler, L. (2001). Novice programming comes of age. In D. C. Smith & A. Cypher (Eds.), *Your wish is my command: Programming by example*. San Francisco, CA, USA: Morgan Kaufmann Publishers.

Smith, A. (2009). Visual perception skills testing: Preliminary results. In *Proceedings of the 3rd international Conference on Tangible and Embedded interaction (TEI '09)*. Cambridge, UK, 16-18 February 2009, pp. 207-208. New York, NY: ACM.

Steinmetz, J. (2001). Computers and *Squeak* as environments for learning. In *Squeak: Open personal computing and multimedia.* New Jersey, NJ: Prentice-Hall.

ToonTalk (undated). *ToonTalk* -- Making programming child's play. http://www.toontalk.com/

Wyeth, P. & Purchase, H. C. (2003). Using developmental theories to inform the design of technology for children. *Proceedings of the 2003 conference on interaction design and children*. New York, NY.

Wyeth, P. & Wyeth, G. (2001). Electronic blocks: Tangible programming elements for preschoolers. In *Human-Computer Interaction - INTERACT'01*. Amsterdam, Netherlands: IOS Press.

Wyeth, P. & Wyeth, G. (2008). Robot building for preschoolers. In U. Visser, F. Ribeiro, T. Ohashi & F. Dellaert (Eds.), *RoboCup 2007: Robot Soccer World Cup XI*, pp. 124-135. Berlin, Germany: Springer.

Zuckerman, O. & Resnick, M. (2003). A physical interface for system dynamics simulation. In *CHI '03 extended abstracts on Human factors in computing systems*, New York, NY: ACM Press.

Leonel Morgado, Assistant Professor
Escola de Ciências e Tecnologia, Dep. Engenharias
UTAD, Quinta de Prados, 5001-801 Vila Real, Portugal
and
GECAD (Grupo de Investigação em Engenharia do Conhecimento e Apoio à Decisão)
Email: leonelm@utad.pt

Maria Cruz, Associate Professor
Escola de Ciências Humanas e Sociais, Dep. Educação e Psicologia
CIFOP, R. Dr. Manuel Carmona, 5001-801 Vila Real, Portugal
Email: mcruz@utad.pt

Ken Kahn, Senior Researcher, Learning Technologies Group
Oxford University Computing Services, University of Oxford
13 Banbury Road, Oxford, OX2 6NN, England, UK
and London Knowledge Lab
23-29 Emerald Street, London WC1N 3QS, United Kingdom
Email: toontalk@googlemail.com