Natural Visual Programming Languages

Tiago Correia, *tiago@cnotinfor.pt*
Department of Computer Science, University of Coimbra, Coimbra

Secundino Correia, *secundino@cnotinfor.pt*
Cnotinfor, Coimbra

Abstract

Children born in an age of knowledge, have access to much more information than we thought. Even without the capacity of reading or writing they understand information and in most cases create new information, build new knowledge. We face a challenge on how to create environments and tools that allow them to develop skills and create new knowledge, specially the ability to program in a natural way, instead of being stuck in the complexity of the environment.

Integrated Learning Environment (ILE) have been studied and implemented along the last decades. They have been defined (Correia et al.) as an ecosystem whose components are: learners, learning organizers, a system or environment where learners interact, technological structured resources including learning objects, methods and strategies for teaching and learning and a set of values to be clarified on action.

We can consider a Natural Visual Programming Language (NVPL) as an ILE providing at least the following five issues: Stimulus provoking the user to take some meaningful actions; Coaching on what makes sense (or not) on the context of programming; Feedback about all steps; Formalization sets; Generalization.

To achieve these goals concepts like the Flow theory (Csikszentmihalyi), Mini-languages (Brusilovsky et al.) and Edutainment (Resnick), should be taken into account. NVLP should allow the users to build computer programs in a natural fluid way and should lead, those who are really interested, to learn the principles, structures and algorithms of programming.

Keywords

Programming; games; visual programming; learning context; integrated learning environment

Introduction

For designing and developing successful products “companies need a development philosophy that targets the human user, not the technology” (Norman, 1999).

In our days technology is everywhere. Children since they born, deal with technology. There are so many devices in our daily life with technology embedded, that we don’t even realise how important they are. TV, mobile phones, microwaves, stoves, clocks, consoles, computers and cars are just some examples of artefacts that we use without even thinking about all the technology that it is behind them. For a child is completely normal to have a mobile phone to talk with friends, or even use the Internet to do it, either by using instant messages or VoIP.

Children born in this new age of knowledge, have access to much more information than we thought. Even without the capacity of reading or writing they understand information and in most cases create new information, build new knowledge.

We assist to a general and deeper trend of computer based technologies imbibed on every day objects by disguising and getting almost invisible. This drift implies new artefacts with imbibed technology and a new functional design much more centred on human being than technology. On this sense we may be talking about disappearing technology, because technologies go behind like electricity that we only notice when it fails. This implies the emergence of new artefacts centred on individual’s daily needs, artefacts with almost hidden computational and information processing power, natural leading to the disappearing of computers like we know them nowadays (Norman, 1999).

What steps do we need to take in order to design and develop such type of environments to allow children to create new knowledge, specially the ability to program in a natural way, so they are not stuck by the complexity of the environment?

Before trying to answer the previous question, let’s first clarify some related overlapping concepts like Constructionism, Integrated Learning Environments and Learning Contexts.

Constructionism is a theory of learning and education based on two different senses of "construction." First, constructionism is grounded in the idea that people learn by actively constructing new knowledge rather than by having information "poured" into their heads. Secondly, constructionism asserts that people learn with particular effectiveness when they are engaged in constructing personally meaningful artefacts (such as computer programs, animations, or robots). Social constructivism adds to this the important role of pairs on this process, by sharing, interacting and joining efforts for building a solution. Metacognition is also enforced by the need of explain our vision to the others.

A close concept linked to constructionism is the concept of microworld forged by Marvin Minsky and Seymour Papert. For them Microworld implies an environment of discovery, where the learners can operate, manipulate and create objects by testing the effects they produce each other. Usually these kinds of environments inlay some form of mathematical processing, allowing users to observe how there decisions affect the organization and balance of the environment.

A microworld was initially more like a learning context or a framework where learners could make theirs constructs, by opposition to digital contents or learning objects. During the nineties and first decade of twenty one century a big emphasis was put on creating and indexing digital contents and learning objects by opposition to microworlds and mind tools for learners to rebuild there on knowledge by doing and interacting.

Nowadays it begins to be consensual the need of balance between learning contexts and learning contents or learning objects. Neither it is viable a pure learning context without contents and coaching, nor it’s effective a pure contents repository. In fact, contents need a context and strong behind models to become valuable and flexible and contexts need content to get meaningful.

In our view an idea that seems to integrate all these concepts is the idea of Integrated Learning Environment (ILE). We can define an ILE as an ecosystem whose components are: learners, learning organizers, a system or environments where learners interact, technological structured resources including learning objects, methods and strategies for teaching and learning and a set of values to be clarified on action.

ICT easily fall on the trap of existing educational structures by a process of domestication where the learning construction process glided to a contents transmission process, although very well packed in wonderful multimedia packages or web based contents repositories. Biodiversity, contexts, values clarification and the empowerment of learners (both students and teachers) subtly glide to unilateral control and one-dimensional views. Networks, learning communities, communities of practice, collaborative contexts easily glide to supervised webs that confine creativity, change and critical thinking.

The challenge of constructing ILE with a strong technological soundness mainly consists on creating learning contexts. By them, learners use materials, concepts and imbibed methodologies to build, destroy and rebuild knowledge synapses in a collaborative, creative and critical way, promoting reflection about the cognitive, emotional and setting values processes in the relation between learners/ groups/ contexts/ contents and containers. In this vision teachers are essentially learning contexts and contents organizers. They should be leaders capable to create values based shared visions. Teachers should be strategists capable to identify opportunities and building bridges, energizing individuals and groups, focusing them on the design of the future. In fact the best way to prepare and predict the future, as Alan Kay says, is to invent it. This is the central role of schools: to invent the future.

We can see a Natural Visual Programming Language a special case of Integrated Learning Environment providing at least the following issues:

- Stimulus provoking the user to take some meaningful actions; these stimulus could be visual and audible. The possibility of kinetic stimulus should also be considered.
- Guidance on what makes sense and makes not sense on a particular context – programming; This should be like a coach or a tutor that gives contextualized guidance on what to do next.
- Feedback about all steps so the learner knows what makes and makes not sense when building a particular computer program (construct).
- Formalization sets, this means the possibility to have an overview of all the steps and the way they interconnect each other, and why the all construct makes sense on this particular context.
- Generalization, which means the possibility to try a similar solution on a different computer program (construct).

Programming

What it is programming? Wikipedia presents a nice definition of programming “Computer programming (often shortened to programming or coding) is the process of writing, testing, and maintaining the source code of computer programs.”

But computer programs are not only what is normally known as application software, like OpenOffice, FireFox or Skype. Computers programs are combinations of instructions to perform a task by the computer. Using different combinations of the same instructions, might lead to different tasks and different results. These instructions are the building blocks of any computer program. Instructions are usually text. An example in C programming language:

```
while (i < 10)
  printf("%d", i++);
```

Figure 1. C programming language example

The instructions presented might not be easily understood by most of the adults that didn't have any contact with programming languages.

For a long time programming was a skill that not everyone was able to master or even learn. The programming languages were very tied with the hardware and were very difficult to learn. Computers kept evolving and so it did the software, that become more user centred. Programming language started to be more abstract from hardware. Many programming languages appeared and new paradigms of programming also emerged. Even so, programming is still a problem for most computer science education institutions.

Papert (1980) described the design of the LOGO programming language as taking the best ideas in computer science about programming language design and "child engineering". It seems that LOGO provided the jump to bring programming to the ordinary user. But now computers have evolved and software is becoming easy to use. Programming languages have also grown to a new level of possibilities and facility, but still are difficult to learn and master (Kahn, 1996).

There have been many efforts to develop programming languages to support the initial steps in the world of programming (Mendelson et al., 1990). Let us to have an overview.

Mini-languages

Mini-languages are a type of microworlds that are designed to be small and simple programming languages for students to learn programming by usually controlling an actor, either virtual or physical device (Brusilovsky et al., 1997). The objective of mini-languages is to provide a solid background for computer science learning by using a context designed for this purpose. Their syntax and semantics are small and simple, the student can spend less time on learning the language and more time in solving algorithms.

The turtle geometry subset from the LOGO programming language provides a mini-language where the student controls a turtle in a virtual world. Usually this subset is as the starting point in learning programming using LOGO. But programming develops general thinking skills as Papert and others (1980) observed, like problem decomposition, component composition, explicit representation, abstraction, debugging and thinking about thinking. To learn to programm is not just a simple skill, but an important one that can boost many others. "It is the acquisition of algorithmic thinking as an explicit, familiar and powerful tool" (Brusilvsky et al., 1997). Many of these skills provide the basis for logical and abstract reasoning that is fundamental to the learning process. By these reasons, may be programming should be explored since the early stages of education.

Mini-languages provide an adequate learning context for programming. They are visually intuitive, simple and a powerful way to introduce students to programming. But mini-languages can be used for many others purposes then programming. Like the LOGO subset could be used to learn geometry. Another example is SOLO programming language (Eisenstadt, 1983) that was used in the context of psychology. It is not very visual appealing for our current standards, but for the time it was developed it was quite evolved. It even had a help/tutoring system to help the students.

For very young children mini-languages can be provided in a different way by using a tangible object. The Robot Roamer is an artefact that uses a subset of LOGO language to control the robot. The interface is a concept keyboard on the shell of the robot with commands to move the robot and create music. The extended version can do more complex tasks as controlling motors, lights and sensors. The context in which the robot is used is very different from learning to

programming, even if the student develops programming skills by using it. The student can change the look of the robot to a context where he feels comfortable.

Robot Roamer is a nice example how mini-languages can be used to learn programming and develop many others skills like creativity, music, electronics, manual dexterity, spatial orientation, geometry, artistic expression, social skills, collaborative work and thinking about thinking.

Games and programming

If we take a look at the Entertainment Software Association information about 2006, 69% of American heads of households play computer games. Games are part of our live and most of the students have already played a game. An extensive research about games with children and adults in anthropology, psychology and education shows that games are important mediators for learning and socialization during life (Ribier, 1996). Computer games have been used for learning for quite a long time, for many different purposes and in many different contexts (Mitchell and Savill-Smith, 2004).

To use them to promote the learning process of programming has also been tried by making the students creating their own games (Leutenegger and Edgington, 2007) using ActionScript programming language as a starting point. One of the reasons to use Flash is that it provides instant visual feedback for the student which helps him to see if the program is correct. Flash development environment is very general and it has not by default a proper context for games development, but by providing snippet code for the student it provides a better learning context for programming games.

There are many environments for programming games, but most of them are not easy to use or are very easy but don't allow building more complex games. They usually lack of the simplicity and smallness of the mini-languages.

But games have many features that make them appealing. Understanding these features and implementing them on software for learning to programming gives birth to a new generation of software. ToonTalk is an example, that such type of software is possible to build (Kahn, 1999).

Flow theory

Mihaly Csikszentmihalyi in an effort to explain happiness introduced the concept of Flow, which has since become fundamental to the field of positive psychology. Flow represents the feeling of complete and energized focus in an activity, with a high level of enjoyment and fulfilment.

During the Flow experience, we lose track of time and worries, which is very common on players. They are completely focused in the performance and pleasure of the game. But it can also be associated with professional athletes and artists, or every human being (Chen, 2007).

Csikszentmihályi research identified eight major components of Flow:

- A challenging activity requiring skill;
- A merging of action and awareness;
- Clear goals;
- Direct, immediate feedback;
- Concentration on the task at hand;
- A sense of control;
- A loss of self-consciousness; and
- An altered sense of time.

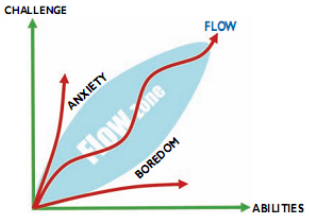

Figure 3. Flow zone factors (Chen, 2007)

A good game keeps the player in the flow zone. This is the challenge of the game designers. Resnick (2004) states that often designers, teachers and parents try to make things "easy" for gamers, learners and children, thinking that they are attracted easynees. But Mihaly Csikszentmihályi has found that people become most deeply engaged in activities that are challenging, but not overwhelming. Similarly, Seymour Papert has found that learners become deeply engaged by having "hard fun". In other words, learners don't mind activities that are hard as long as the activities connect deeply with their interests and passions.

Edutainment

By combining games and learn, the term edutainment nurture in our head. Edutainment attempts to merge both concepts: education and entertainment. As previously said, content should not be provided without a learning context, but the entertainment industry is providing a large amount of content as edutainment. Knowledge can't be provided as content. More sceptically knowledge can't be provided anyway; We only can provide contents, hopefully on a real and correct learning context.

Resnick (2004) distinguish edutainment from "playful learning" since the focus is on the play and on the learning, and not on the entertainment and education. We also agree with Resnick that such combinations of games and programming languages should provide new software where the player/student have an active role playing and learning simultaneously.

NVPL offer the right contexts to reach formal thinking, using a transparent and natural environment.

Visual programming languages

Currently there are many visual programming environments. Wikipedia presents a good definition of Visual Programming Languages (VPL), "A Visual programming language (VPL) is any programming language that lets users specify programs by manipulating program elements graphically rather than by specifying them textually." Using graphics to represent programming blocks provides a better efficiency and simplicity on the interaction between the user and the computer.

Designing a VPL for creating any type of computer program, is a very difficult task. Visual programming is a good way to start to learn to program, to create models with some level of abstraction or to design mock-ups of applications. It would be very hard to program the kernel of an operating system using a VPL, since this type of programming is very tight with the hardware.

Some examples of VPLs are StarLogo TNG and Scratch . Both environments have many similarities, but are designed to be used in different contexts. StarLogo TNG focus on the learning of programming using mostly LOGO syntax and programming rules, while Scratch is on expression of music, dance and arts, but it can also be used in the same context as StarLogo TNG. A computer program is not written in these environments. Instead blocks with different shapes must be combined with each other in a jigsaw way. It is not possible to have syntax errors which are very common in the textual programming language (a missing semi comma, misspelling a command). Many common programming errors of mismatch type variables don't

happen on these environments, since the block of different types have different shapes and it would not fit on wrong blocks.

Many environments are in fact VPL, but they are just the same instructions that exist on regular programming languages wrapped in images with different shapes and colours. Even so they have the advantages stated before, so they are a preferred media to learn to program. But, do the shapes and colours have a meaningful sense for the user? That is another question, which is not going to be addressed on this paper.

Natural Visual Programming Languages

VPL in most cases can not be considered mini-languages, but as most of the mini-languages, despite the fairly intuitive interface, they can not be used in a natural way. The *reacTable* is a very good example on how electronic music can be built naturally, with a very small learning curve. It is an interface that requires zero manual, zero instruction. Any user can create electronic music (Jordà, 2003). The *reacTable* is sonically challenging and interesting, learnable, suitable for complete novices and for advanced electronic musicians and totally controllable (Jordà, 2003). The same concept could be applied to NVPL, making the use of such environments natural for the user.

Natural Visual Programming Languages (NVPL) would still require time to learn to program, as in the *reacTable* it would require time to learn to build *real* music. But the initial learning curve to use the *reacTable* needs no tutors, tutorials or manuals. The interface is so clear and self-explained that the user easily grasp how to use it. The visual response of the application to the user is so intuitive that can be used by any user naturally. The feedback (visual and sound) provided completely guides the user on how it should proceed (Jordà, 2003). Also, when an object is added to the table, there is an immediate visual and sound feedback of the consequences of adding it.

By using such an environment does not means that the user will be able to build music, but that can do it much more easily. Since the usage of the environment is not any more an obstacle, users can be focused by the learning context to learn musical meaningful contents, by doing. In Mindstorms, Papert (1980) point that children learn a lot on their first years of live, without being able to read or write. They do many mistakes, but if they don't experiment, they don't learn.

In ours days we tend to think that someone unable to read and/or to write can not learn most of the knowledge available, which is not true. Literacy does provide an enormous boost in the possibilities of learning, since we don't depend on the help of others, but there is much more knowledge besides the ones provided by text. Our body has 5 senses, and if all of them can be stimulated, different perspectives of the same information are provided to us, allowing us to create a much more accurate and quickly knowledge. Learning styles and multiple intelligences hypothesis by Gardner also reminds us that using only logical and linguistic reasoning is not the best trend for everyone to learn.

NVPL should become environments where anyone can build computer programs by combining graphical elements, either by using mouse, gesture, voice or any other device, or even by using tangible objects. The main difference is that formal literacy should not be essentially required to begin programming in these NVPLs. In an ideal environment even users with disabilities or autism should be able to use them. Not only visual interfaces should be provided to "read" information, but sound and touch should also be included to stimulate a richer experience to the user. "Good animation and sound effects help greatly in making an environment self-revealing" (Kahn, 2004). The Flow theory by Csikszenthmihalyi give us an idea about how this environments should challenge the user with clear goals, but without loosing the sense of control and by getting immediate feedback on every action (or even while no action is taken).

Kahn (2004) states it is very difficult to create a completely self-revealing environment. So to help creating a self-revealing environment the user must be coached on the right way (natural way) and in context; more possibilities to fulfil it purposes would emerge on the context of

programming. Indeed it's impossible to learn without stimulus and without guidance. If we are kept on a empty room with no doors neither windows, our learning will be very limited. As the feral children case studies show us the right kind of early interaction is essential for normal development and learning (Candland, 1993). As much richer and contextualized are the stimulus more possibilities we have for learning. This richness and significant context can be mediatised by our parents, teachers, companions, but can also be a book, a virtual companion or a screen computer that makes sense for us and provokes a learning path where we build some meaningful construct for us. This means that we can connect what we are learning with ours previous learning experiences and that this make sense for us.

There are not yet NVPLs softwares. Dragões e Companhia and ToonTalk are some examples of uncompleted NVPL. And they are not prepared natively to be used by people with disabilities. Also the learning process to master these environments is not short. Even thou, these environments use a very different approach to VPL, so we will make an overview of both next.

Dragões e Companhia

Dragões e Companhia is the second generation of the software Floresta Mágica produced by Cnotinfor. The new generation follows the same principles of the previous, but with enhanced engine and sleek graphics.

The main idea for programming is to create rules for objects that are created on a page. The objects are always visual objects (frog, donkey, warrior, car, apple …). Each object has it own rules which are created by a condition (event) and a sequence of actions that should be performed. There are special building blocks (control stones) that can be put alone in a rule to making the object be controlled by a device like a mouse, keyboard and/or joystick.

The metaphor used for the building blocks is that they are stones that should be put together in a rule, which is a papyrus. An object can have many rules – papyrus. It is an odd metaphor, but the children that used it never thought it has a strange one (Andrade, 2002).

This environment provides tools for building simple and complex games, stories, simulations and animations. The building blocks are 14 events/conditions, 28 actions and 5 controls stones. It seems a small number, but this low number provides a mini-language for systematic problem solving. Since it was designed for very young children (4 to 14 years old) the interval of the parameters of the stones is limited.

The design idea for the engine of Dragões e Companhia is that there are events occurring, and the user must create actions to respond to these events, that can trigger other events.

A nice example of a simulation that can be programmed is two kids (objects) playing with each other and they throw a ball to each other. This same simulation, as almost any computer programs can be programmed in many different ways. A possible solution is to create a rule to each kid (object) that when a red ball it him, throw a red ball.

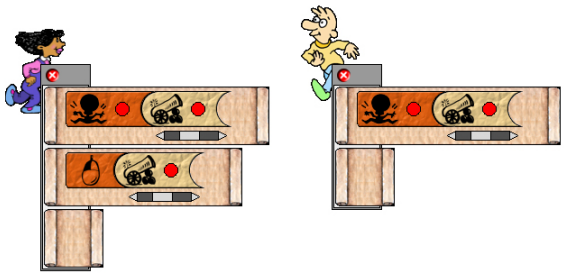

Figure 4. Rules of objects in Dragões e Companhia

There is an extra rule on the girl, when the girl is clicked throw a red ball. This rule is needed to start sending the ball between the two kids.

One of the goods things in favour to become an NVPL is the fact of providing challenging starting points, but without a tutor it is not easy for most of children to engage and explore the starting points in a productive way. Also it provides a very concrete way of representing the constructs that allows the user to think about the solutions created to solve a problem.

Dragões e Companhia has some problems to be NVPL. The interface is not intuitive, and in the firsts approaches it takes some time to get used to the method of creating computers programs. It also lacks of some visual and audible responses on the user interaction. When user first enter in the environment it is not clear what he can do. A blank page is shown and naturally he asks: what can I do? The only thing possible is trying the handles. But by clicking on the handles it's not clear what it is possible to do. Dragões e Companhia should have a mechanism of contextualized coaching in order to approaches it from the NVPL concept. It should offer more stimulus and guidance for building meaningful constructs. When we do things that have no sense, the environment should give advice and feedback pointing to other possibilities that make sense on the context.

It also lakes of the possibility to create new building blocks, which in many cases would help the children to decompose problems in smaller problems. The possibility to extend the current language is one of the major features of programming languages, but Dragões e Companhia lakes of it. So to have a small number of blocks, and not being able to extend them, it is good for a quick learning, but not good in the long run, since children will have more difficulty to express them selves, since only a small amount of blocks are available. Only children more motivated and with a better mathematical knowledge will learn to use them beyond the trivial purpose (Papert, 1980). But once learned it allows the user to build very quickly and easy, many graphically appealing computer programs.

ToonTalk

ToonTalk provides a complete different metaphor for programming from what computer scientists are used to. The programming is done by demonstration in a virtual animated interactive world. All environment objects are familiar to the user. Birds, scales, trucks, robots, vacuum, notebooks, and others. One of the most interesting features is that the source code of the computer program is animated. For very young children this is great, since the objects behave just like ordinary objects. For example, if the user gives an object to a bird it will take it to his nest.

All the objects in ToonTalk have a corresponding computational concretization (Kahn, 1996):

Computational	**ToonTalk**
Computation	city
agent (or actor or process or object)	house
methods (or clauses or program fragments)	robots (with thought bubbles)
method preconditions	contents of thought bubble
method actions	actions taught to robot inside thought bubble
tuples (or arrays or vectors or messages)	cubbies
comparison tests	scales
agent spawning	loaded trucks
agent termination	bombs
constants	number pads, text pads, pictures
channel transmit capabilities	birds

channel receive capabilities	nests
program storage	notebooks

ToonTalk has a tutorial mode called Puzzle, that teaches many features and methods of usage of ToonTalk, by solving a series of problems that are part of a story. Each problem presents a new feature/tool of ToonTalk. By presenting the tools/features gradually in a predefined order, make it possible to master ToonTalk. Each tool is presented, as it is needed by the user to solve the problem. Similar and very good examples of the success of such approach are computer games like Lemmings or The Incredible Machines. In such games each new tool and ways of solving problems are presented gradually, and only when the user is able to use them, new ones are presented to him. In this way ToonTalk presents most of the features to the user in a contextualized environment (adventure game) and in a gradual form.

During the normal usage of ToonTalk, Marty provides tips and tricks to help the user to use ToonTalk. Also when something is not possible to do, there is a feedback either visual, audible and/or sensorial (if a force feedback joystick is connected to the computer).

ToonTalk is not an environment that a user can naturally use without tutoring, although it has a character called Marty that is always present (if the user wants it), to give feedback about the tools and objects. It also gives some hints on the beginning on how to use every object and tool.

ToonTalk provides modes that could with some evolution, convert it into a more natural environment. The Puzzle mode and the demos are very important modes for learning programming languages, since they provide ways for the user to learn how to use them. But demos must be carefully produced since the current standards of videos are the ones presented today in TV and cinema, with good voice, script and graphically seductive (Kahn, 2004).

One very important characteristic of ToonTalk is that it was designed having social learning in mind. It fulfils this purpose by several ways, but the more interesting ones are: working with a long viewing distance (great to use in classroom contexts), multiple players support and network collaboration without losing its metaphor (long distance birds) (Kahn, 2004).

Conclusion

Natural Visual Programming Languages (NVPL) is still a concept and there is a long path to walk to develop such environments. ToonTalk it seems one of the softwares that more resembles with NVPL.

Creating environments that combine so many features of so many different software types is the next step. ToonTalk might provide the correct framework to build this type of software, but the programming paradigm is very different from what is usually learned and used. Combining the engine of Dragões e Companhia with the learning contexts provided by ToonTalk might be another option to explore.

We believe that this kind of software will revolutionise the way programming languages are thought and that the learning context around this subject would change. The challenge now is to find feasible solutions for each characteristic of NVPL that can be combined in one single solution.

References

Andrade, M. (2002) *Floresta Mágica: o contributo dos parceiros na construção do Playground*. Dissertação de mestrado. Lisbon: Universidade Aberta.

Brusilovsky, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A. and Miller, P. (1997) *Mini-languages: a way to learn programming principles*. Education and Information Technologies (2), 65-83.

Candland, D. K. (1993) *Feral children and clever animals: reflections on human natu*re. Oxford University Press. Relates the story of Kamala and Amala.

Chen, J. (2007) *Flow in Games (and Everything Else)*. Communications of the ACM 50 (4), 31-34.

Correia, S. et al. (2004) *Micromundos AIA*. Coimbra: Cnotinfor.

Csikszenthmihalyi, M. (1990) *Flow: The Psychology of Optimal Experience*. London: Harper Perennial.

Jordà, S. (2003) *Interactive Music Systems for Everyone - Exploring Visual Feedback as a way for creating more intuitive, efficient and learnable instruments*. Stockholm Music Acoustics Conference (SMAC 03), Stockholm (Sweden).

Kahn, K. (1996) *ToonTalk - An Animated Programming Environment for Children*. In *The Journal of Visual Languages and Computing*, Volume 7, number 2, June.

Kahn, K. (1999) *A Computer Game to Teach Programming*. National Educational Computing Conference 1999.

Kahn, K. (2004) ToonTalk - Steps Towards Ideal Computer-Based Learning Environments. In Tokoro, M. & Steels, L. *A Learning Zone of One's Own: Sharing Representations and Flow in Collaborative Learning Environments*. Ios Pr Inc.

Leutenegger, S. and Edgington, J. (2007) *A Games First Approach to Teaching Introductory Programming*. SIGCSE'07, 115-118. Covington, USA

Mendelson, P., Green, T. and Brns, P. (1990) Programming languages in education: the search for an easy start. In Hoc, J., Green, T., Gilmore, D. & Samway, R. (eds) *Psychology of Programming*, 175-200, London:, Academic Press.

Mitchel, A. and Savill-Smith, C. (2004). *The use of computer and video games for learning*. London: Learning and Skills Development Agency.

Norman, D. (1999) *The Invisible Computer*. London: MIT Press

Papert, S. (1980) *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books.

Piaget, J. (2001) *Studies in Reflecting Abstraction*. Hove, UK: Psychology Press.

Resnick, M. (2004) *Edutainment? No thanks. I prefer playful learning*. Associazione Civita Report on Edutainment.

Rieber, L. P. (1996) Seriously Considering Play: Designing Interactive Learning Environments Based on the Blending of Microworlds, Simulations, and Games. Educational Technology Research & Development. Georgia: The University of Georgia.

Wikipedia. (2007) *Computer programm*ing. Retrieved March 10, 2007 from http://en.wikipedia.org/wiki/Programming.

Wikipedia. (2007) *Constructionist learning*. Retrieved April 2, 2007 from http://en.wikipedia.org/wiki/Constructionist_learning.

Wikipedia. (2007) *Programming language*. Retrieved March 10, 2007 from http://en.wikipedia.org/wiki/Programming_language.

Wikipedia. (2007) *Visual programming language*. Retrieved March 10, 2007 from http://en.wikipedia.org/wiki/Visual_programming.