Programming Without a Computer: A New Interface for Children under Eight

Peta Wyeth and Helen C. Purchase
Department of Computer Science and Electrical Engineering
University of Queensland, Australia
peta@csee.uq.edu.au
hcp@csee.uq.edu.au

Abstract

Electronic Blocks are a new programming interface, designed for children aged between three and eight years. The Electronic Blocks programming environment includes sensor blocks, action blocks and logic blocks. By connecting these blocks children can program structures that interact with the environment. The Electronic Block programming interface design is based on principles of developmentally appropriate practices in early childhood education. As a result the blocks provide young children with a programming environment that allows them to explore quite complex programming principles. The simple syntax of the blocks provides opportunities for young children unavailable through the use of traditional programming languages. The blocks allow children to create and use simple code structures. The Electronic Block environment provides a developmentally appropriate environment for planning overall strategies for solving a problem, breaking a strategy down into manageable units, and systematically determining the weakness of the solution. Electronic Blocks are the physical embodiment of computer programming. They have the unique dynamic and programmable properties of a computer minus its complexity.

1. Introduction

Electronic Blocks aim to provide young children, aged between three and eight years of age, with a new interface for exploring programming concepts. Electronic Blocks represent a paradigm shift, away from using the computer to teach programming, towards a physically embodied system that provides programming experiences that involve active manipulation and transformational of real materials. This programming interface has been designed to take into account the special needs of children under eight and has been based on early childhood development and learning research.

Electronic Blocks are blocks with electronic circuits inside them. By placing electronic blocks on top of one another young children build the equivalent of "computer programs". Sensors and effectors built into the blocks allow children to build structures that interact with the environment. The sensor blocks detect *light*, *sound* and *touch* while the effector or action blocks are capable of producing *light*, *sound* and *movement*. By connecting sensor blocks with action blocks children can program their own structures. They might create a group of "robots" and these robots might wink at each other, move when they hear sound, or be capable of following a light. There are also logic blocks which logically *negate*, *delay*, *and*, and *toggle* signals between sensor blocks and action blocks. The inclusion of these blocks adds an additional dimension to the capabilities of the children's creations.

The design of these blocks has now been completed and the implementation phase of the Electronic Block project is nearing the completion. Once successfully implemented, the Electronic Blocks will be extensively evaluated. This evaluation has been designed to explore the extent to which electronic blocks allow children to explore programming concepts as hypothesised in this paper.

1.1. Learning to Program: A Powerful Educational Experience

When computers were first introduced in early childhood settings, there was a propensity for using the computer as a tool to reinforce existing practices, facilitating activities such as reading, writing and mathematics. This mechanistic style of computer use, still seen to some extent in schools, led educators to ask whether or not computer-based activities offer anything that is substantially different from what can be obtained in the classroom by other means [13][3][14]. The answer for some has been for educational institutions to embrace computer use that offers opportunities for children to explore the computer's unique dynamic and programmable properties. From as early as 1980, with the ground-breaking work of Papert [9], researchers recognised that computer programming as an educational activity had great potential as a vehicle for the acquisition of useful

cognitive skills such as problem solving and reflective thinking.

The research outlined in this paper follows this school of thought. It is widely acknowledged that while computers can have many functions within an early education setting, its power as a tool for technology education lies in the programmable and dynamic properties unique to it. It is these properties which make it different from other media with which children interact. This type of computer exploration allows children to become involved in technology in a way recommended by Raizen et al. [10]. Computer programming requires children to use their intellectual resources while involved in processes such as designing, producing and using, to create systems and structures. Programming also provides the opportunity for children to become involved in "seeing and constructing in terms of new formal knowledge of how things are put together" [7].

For over a decade researchers a MIT Media Laboratory have been studying the richness of experiences that involve exploration of the dynamic and programmable properties of a computer. The strong belief that children benefit greatly from becoming creators, not just consumers, of computer activities, drives their research. It is a philosophy embraced by the research discussed in this paper. Programming a computer allows children to have an impact on the technology they are using, they become the creators, and they are in control. Through learning to program a computer children develop a much deeper relationship with, and consequently deeper understanding of, the computer [12].

Unfortunately for young children, the ideas about instructions and sequence, which form the core of programming, are not necessarily simple. At a time when young children are only just acquiring the rudiments of notational systems and are struggling with symbolisation in language [13], it would be unreasonable to expect them to cope with the symbolic systems required to successfully program a computer. In addition, programming languages are artificial rather than natural languages and consequently, they have a different epistemology that deals with the unfamiliar world of computer data structures and algorithms [15]. This makes them even more difficult to learn.

As an alternative to the computer, Electronic Blocks are designed to provide preschool and early primary school children with a resource that has the unique dynamic and programmable properties of a computer minus its complexity. Electronic Blocks are the physical embodiment of computer programming. They are naturally less complex to program than a computer as they do not require a knowledge of complex symbolic systems.

2. Developing a New Programming Interface

The Electronic Block programming interface incorporates two sets of design criteria, those that ensure the age-appropriateness of the design and those that capture the programmable elements of a computer.

2.1. Design Criteria

In order to meet principles of developmentally appropriate practices that reflect current knowledge and beliefs about what constitutes high quality, developmentally appropriate early childhood education, the design of Electronic Blocks must ensure:

1. activities are open-ended and discovery-oriented, allowing children to be actively involved in the learning process;
2. interaction encourages child-initiated play;
3. experiences involve active manipulation and transformation of *real* materials;
4. entry level knowledge and experience is kept to a minimum;
5. provision is made for children's varied skill and ability levels;
6. construction activities that involve design, creation and evaluation processes form the basis of interactions.

(based on Bredekamp and Copple [2])

Applying the principles of developmentally appropriate practices, aims to ensure that the Electronic Blocks are a developmentally appropriate resource for providing programming experiences to children under the age of eight.

In addition, in order to create a *programmable* resource, electronic blocks have been designed using guidelines outlined by Resnick [11]. These guidelines have formed the basis of the following design criteria. Electronic Blocks need to:

7. be non-algorithmic – the path of action is not fully specified in advance;
8. be complex – the total path is not visible;
9. incorporate uncertainty – not everything that bears on the task at hand is known;
10. allow users to find structure in apparent disorder;
11. yield multiple solutions, each with costs and benefits.

These 11 design criteria have formed the platform on which the Electronic Blocks have been based.

3. A Description of Electronic Blocks

The Electronic Blocks have been designed so children can connect them just as they would any other blocks. The blocks have been made by placing electronics inside Lego

Duplo Primo™ blocks. This ensures that the blocks are easy to stack and connect.

Electronic Blocks have inputs and outputs and when connected, the output of one block controls the input of another. There are three kinds of electronic blocks: *sensor* blocks, *action* blocks and *logic* blocks. Sensor blocks detect light, touch and sound within the environment. Action blocks produce some kind of output: a *light* block produces light, while a *movement* block is capable of motion. The sensor blocks will provide the signals that make the action blocks "act".

Logic blocks have an intermediary role. Placed between a sensor block and an action block they have the ability to alter the expected action.

3.1. Sensor Blocks

There are three Electronic Sensor Blocks: a *seeing* block, a *hearing* block and a *touch* block. These blocks are capable of detecting light, sound and touch, respectively. They are single connector blocks that have an input attached to the upper connector and an output attached to the lower connector (see Figure 1). The input is off unless it explicitly receives an on signal. The input and the sensor are ORed together to produce the output. As a result when two or more sensor blocks are stacked on an action block any sensor input will trigger the action block.

3.2. Action Blocks

Action blocks produce some kind of output. The *light* block produces light, the *sound* block produce sound and the *movement* block is capable of motion. All action blocks are double connector blocks (see Figure 1). They are physically constrained by a base plate so that they cannot be placed on top of another block and have to be positioned at the bottom of a block stack. The two upper connectors may receive an input signal. Each action block works as an OR – the action takes place if the block receives input from either upper connector.

3.3. Logic Blocks

With the exception of the *and* block, these blocks are single connector blocks with an input attached to the upper connector and an output attached to the lower connector (see Figure 1). The input of the *not*, *toggle*, and *delay* blocks is off unless an on signal is received. An on signal will produce an output relative to the logic of the block.

The *and* block, a double connector block, has two upper connectors which may receive an input signal (see Figure 1). The block works as a logical AND – it must receive an input from both connectors to produce an output. The output signal produced is attached to both lower connectors.

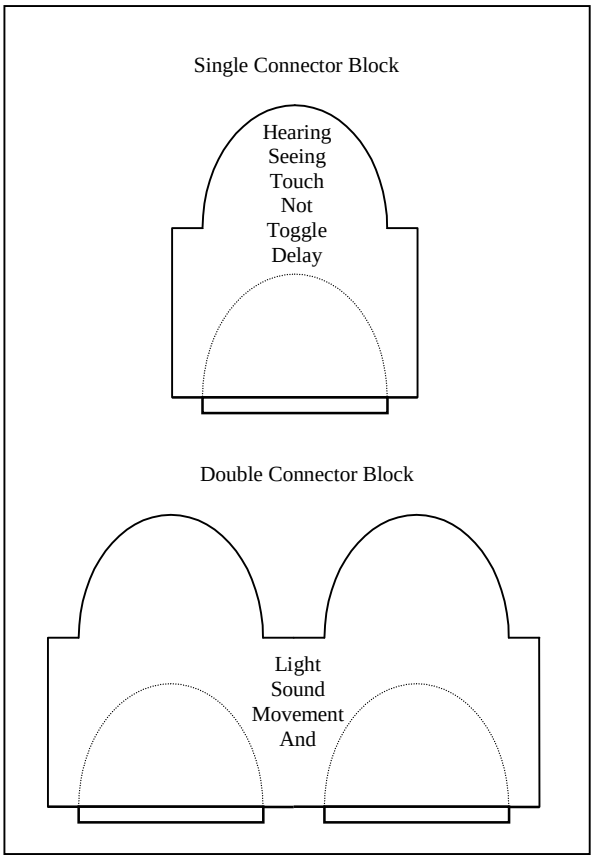


Figure 1. Single and double connector electronic blocks

4. Young Children Programming with Electronic Blocks

Children of all ages are capable of programming using the electronic blocks. They have been designed to be used by children across the early childhood spectrum – children under five who are not engaged in the formal education process as well as school aged children.

4.1. Pre-schoolers Programming ~ simple constructions producing simple behaviours

Even very young children can use a set of Electronic Blocks. Three, four and five-year-olds can stack and balance the Electronic Blocks and as a result become familiar with their simple functionality. Children of this age can experience electronic blocks in the same manner they experience any other construction material. As a result of simply playing with the blocks, children can accidentally produce interesting behaviours that they would undoubtedly find fascinating. They might build a block tower that flashes when they talk, or moves with

their touch. These are examples of simple sensor-action combinations. Given a set of three sensor blocks and three action blocks, there are a total of nine such combinations. While young children are becoming used to the functionality of Electronic Blocks it may be advantageous to limit interactions to these simple input-output combinations. Logic blocks may be added when children demonstrate confidence using the sensor and action blocks.

4.2. The Impact of Logic Blocks

The addition of logic blocks to the set of Electronic Blocks opens up a wide variety of additional construction opportunities. Logic blocks provide users with the capability to:

- only produce an action if two input signals are received simultaneously,
- produce an action if a particular stimulus is *not* received,
- add a time delay between the sensor input and the desired action, and
- toggle the input so that in the first instance the stimulus from the environment will "turn the action on" and the second instance of the stimulus will "turn the action off".

A task that sees the introduction of a logic block is the creation of a car that starts when you clap and stops when you clap again. A *toggle* block placed between a *hearing* block and a *movement* block will achieve this result. Logic blocks add to the complexity and variety of structures that may be created.

4.3. Increasing Complexity – Creating Interacting Electronic Block Stacks

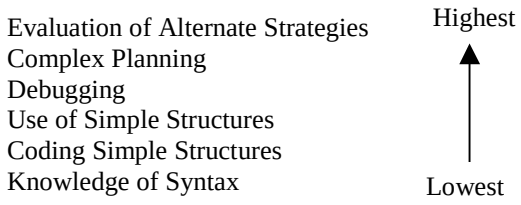
A fascinating aspect of Electronic Blocks is their ability to interact not only with the environment but also with each other. An example of two Electronic Block structures interacting is the creation of a remote control car. By creating one block stack which contains a *touch* block and a *light* block and another stack which has a *seeing* block on top of a *movement* block, a child has effectively created a remote control car. By pressing the *touch* block, the child triggers the light. This light in turn is detected as an input by the *seeing* block which actives the *movement* block.

5. Promoting Programming Skills with Electronic Blocks

Programming is generally thought of in terms of code writing and debugging – in essence consisting of "a series of written instructions that make a computer accomplish a task" (Pea and Kurland, 1984 as cited in [8]). However, in recent years, the changing nature of computer interfaces has lead people to question this definition of programming. Given the development of direct manipulation programming languages such as ToonTalk™ [6] and Cocoa [15], the acceptance of "languageless" programming as a legitimate alternative for the purpose of teaching programming is gaining momentum.

In addition to "writing code" programming is about defining a problem and being able to generate effective programming approaches with an awareness of the alternatives that are most appropriate. A programmer needs to design a solution based on a hypothesis of how the problem will best be solved. Developing a solution requires an evaluation of alternate strategies and careful planning of the steps required to achieve this solution. Subsequent testing is required to determine the strengths and weaknesses of the original hypothesis.

These are the skills that the Electronic Blocks programming environment aim to promote. In analysing the extent to which Electronic Blocks can promote the development of such programming skills the following hierarchy (developed by Oakley and McDougall [8]) has been used:



Each of these skills is examined with respect to how it can be achieved more readily using Electronic Blocks than by the use of a traditional programming language.

5.1. Knowledge of Syntax

The syntax of Electronic Blocks is very simple. In order to produce some kind of behaviour children must

1. Include an action block in the block stack.
2. Include a sensor block in the block stack.
3. Place the action block at the bottom of a block stack.

All action blocks are physically constrained by a base plate ensuring that they cannot be placed on top of another block. Therefore, creating a block stack to produce a behaviour is simple. All that is required is one sensor block and one action block. A child can create a car that moves in a bright environment by placing a seeing block on top of a movement block. By placing a touch block on a light block a child has created a camera flash – the light flashes when they hit the button.

In order to use logic blocks successfully children need to understand that:

1. Logic blocks alter the signal they receive from the blocks above them. If they are at the top of a block stack they will not effect the operation of that stack.
2. If two sensor blocks are placed on top of a logic block then the logic block will perform the logic operation when *either* environmental condition is met. In the example in Figure 2 the light will turn on in the dark (ie. not light) *or* when not touched. Either condition effects the behaviour of the action block.

Logic blocks increase the complexity of electronic block syntax. The order of placement of the logic blocks within a program stack is an important consideration, as each change in order has the potential to yield a different outcome. Fortunately, even with this increase in complexity, two factors make it possible to work through logic issues using trial and error. Firstly, any solution stack that has a sensor block and an action block yields a testable behaviour. This allows the child to easily determine the impact of a logic block in a stack. Secondly, changing a program stack is a relatively straightforward operation – simply pulling the blocks apart and reordering the stack.

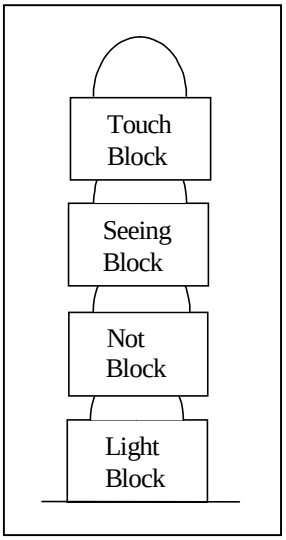


Figure 2. Program stack including a logic block

Compared with the syntax of even the simplest computer programming language, the syntax of electronic blocks are significantly less complex. This lack of complexity ensures that Electronic Blocks provide a powerful platform for teaching children fundamental programming concepts. Reducing the complexity of the syntax provides children with greater opportunities to focus on high-level programming concepts such as debugging, planning and evaluating strategies.

5.2. Coding Simple Structures

An important issue in learning to program is program design – successful mapping between problem domain and program domain (as defined by Brooks cited in [4]). Research indicates that end user programmers have difficulty dealing with entities in the program domain that do not have corresponding entities in the problem domain and an abundance of low level primitives is one of the great cognitive barriers to programming (for a review of this research see [4]). Given the cognitive limitations of young children, it may be concluded that such difficulties would be amplified.

An important consideration, then, for creating a programming environment for young children, is the inclusion of entities that map directly back to the problem domain. Electronic blocks represent such entities. If a child wishes to create a robot that sings in the light, then it is a simple matter of mapping the problem to the blocks available. That child will need to include a *seeing* block so the robot can tell whether it is light or dark and a *noise* block to create the sound. If that child then wanted to create a robot that does *not* sing in the light it is a matter of adding a *not* block to the program stack. If the child wants the robot to sing in the light after a *delay* they just need to add a *delay* block.

This task-oriented style of programming makes the coding of simple structures relatively simple. Each entity in the Electronic Block system maps directly to the problem domain.

While new task-specific visual programming languages are being designed to address this issue, most computer programming languages are full of low-level primitives that do not map directly back to the problem domain. The act of coding a simple structure in most programming languages requires a knowledge of low-level primitives and how they should be combined to achieve the desired result. Given that young children rely more on their visual and auditory perception for knowledge than they do on logical thought processes [5] it is understandable that they find type of programming is extremely difficult.

5.3. Use of Simple Structures

A child can create a camera flash by placing a *touch* block on top of a *light* block. They can create a remote control car by placing a *seeing* block on a *movement* block and then using the camera flash as the remote control (see Figure 3). Pressing the touch sensor would result in a light turning on. This light would be sensed by the *seeing* block, which would in turn trigger the movement of the *movement* block. A remote control vehicle has been created.

This is an example of a way in which Electronic Blocks allow children to reuse simple “code structures”. The same

camera flash may be subsequently used to trigger a series of *seeing* block, *light* block combinations, which grouped together make an interesting lighting display.

Once familiar with the syntax of the Electronic Blocks and experienced in the coding of simple structures, reuse of the structures is simply a matter of locating them near another block stack so they are able to trigger a sensor located on that stack.

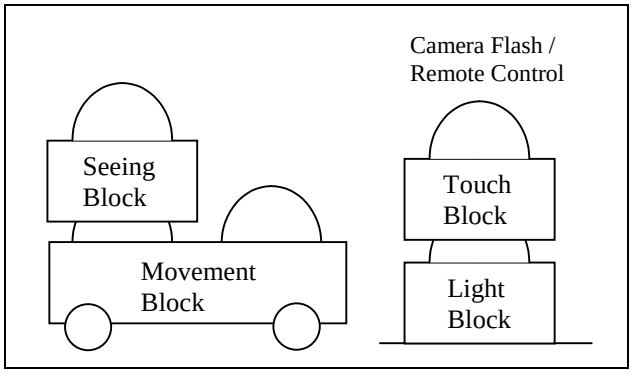


Figure 3. An example of reusing simple code structures

Using simple structures in traditional programming languages is significantly more complex. This is largely due to the greater complexity of both the syntax and semantics of these languages. A new generation of the Logo programming language, Boxer, developed at MIT and designed as the successor of Logo, attempts to address this issues. The designers have created an environment where computational objects such as programs are visual units (boxes) that may be easily manipulated as a whole. They have attempted to create a system that allows children to directly change or use anything put on the screen, thereby creating a simple form of concrete programming (di Sessa, 1984, cited in [1]). Such work acknowledges the difficulties faced by young children when learning to master a programming language.

5.4. Debugging

An important consideration in the debugging process is the need that programmers have to evaluate incomplete programs as well as finished ones [4]. In fact, studies show that the less experienced the programmer, the smaller the amount that is produced before it must be evaluated. An important characteristic of the Electronic Blocks is that small fragments of larger programs are generally complete within themselves and may easily be tested in a stand-alone fashion. An example of this is the creation of a vehicle that moves either when it hears a noise or when it sees a light. A partial solution would involve placing a *hearing* block on a *movement* block. This "code fragment" could be tested simply by making a noise to ensure that the vehicle moves. The addition of a *seeing* block would complete the solution.

The Electronic Blocks provide an environment for novices where it is easy to check a program fragment before adding to it. Every step can be checked individually, as can combinations of steps, to see whether something has gone wrong. This is a programming environment that allows seriously incomplete program fragments to be evaluated. Most computer based programming environments do not allow this flexibility.

5.5. Complex Planning

A programmer must plan the overall strategy for solving a problem, break this strategy down into manageable units, and to systematically seek out weakness in one's reasoning.

The Electronic Block programming environment provides the opportunity for children to become involved in this type of planning activity. To illustrate this point, the example of creating robots that wink at each other is used. Initially a child might create two program stacks, each containing a *seeing* block and a *light* block. The *light* block represents the "wink". This solution has a problem. Neither robot will initiate the winking sequence. Both are sitting waiting for a light input. The solution may involve the addition of a *not* block to one of the stacks. This modification to Robot1 will result in it starting the winking sequence. When Robot2 winks in response, Robot1 will sense the presence of the light and its light will turn off. As a result Robot2's light will then turn off. This results in a wink by Robot1, and the process is repeated. This solution might be further improved by the addition of a delay block to Robot2. This solution is illustrated in Figure 4.

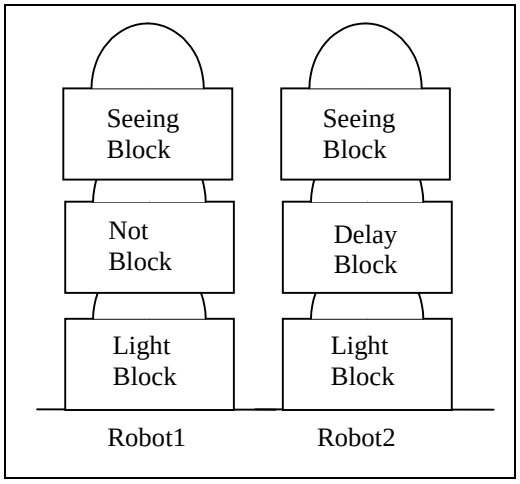


Figure 4. A solution to the winking robots problem

The emergent nature of the problem makes it an ideal programming task for children. The path of action is not fully specified in advance, nor is it visible. Children are required to make a plan of action, evaluate each step through testing their solution and make any necessary adjustments.

Possessing strategies for planning and debugging is a prerequisite for programming success. Many bugs arise as a result of plan composition problems – difficulties in putting the 'pieces' of a program together [4]. Electronic Blocks avoid this difficulty because each piece of a program works independently and each step towards a solution may easily be tested. In addition, research suggests that using spatial reasoning for support may be useful in the planning and debugging processes [4]. Electronic Blocks provide more scope for visual and spatial reasoning than text based languages.

5.6. Evaluation of Alternate Strategies

There are many Electronic Block programming tasks that have multiple solutions, each with costs and benefits. As a physically embodied programming environment, Electronic Blocks provide a powerful means for evaluating alternate strategies. Children, unsure of the best solution, may easily build two solutions concurrently, and then compare and contrast them. While young children might struggle to logically analyse the positives and negatives of alternative strategies (as may be required with a traditional programming language), their natural curiosity and inherent desire to construct their own understanding of the world, acts as a driving force in the exploration of such alternatives with Electronic Blocks. Children are true scientific explorers of their environment and within a non-threatening environment that encourages autonomy, children will naturally seek alternate strategies and test the limits of their creations.

6. Conclusion

Electronic Blocks are physically embodied and consequently will provide experiences that involve *active* manipulation and transformational of real materials. The Electronic Blocks are undeniably a resource that allows children to work both autonomously and actively, and the variety of expressive opportunities they offer guarantees open-ended, discovery oriented learning experiences.

6.1. Successful, Independent Programming

The examples of programming outlined above show that children can use the Electronic Block programming environment *independently* to build programs. The theory of developmentally appropriate practice utilised in the design of the electronic blocks guarantees a learning environment where children can experience programming unimpeded. Alan Kay of the Apple Research Laboratory identified this as of key significance.

> *"One of the things I have been interested in for many years … is the possibility of giving young kids direct access to interesting ideas via an interface that does not require the mediation of adults at all."*
>
> [7]

The Electronic Block programming interface is simple to master and while things might not always go to plan, making adjustments is as easy as pulling blocks apart and rebuilding them.

6.2. A Flexible Programming Interface for Children of All Ages

In designing the Electronic Blocks consideration has been given to providing a number of levels at which children can work. They can be used as "normal" building blocks or provide challenges through the use of logic blocks and through the creation of structures that interact. The flexibility - with respect to skill and ability levels - with which the Electronic Blocks have been designed, makes them a powerful resource for teaching programming concepts. However, it is the more complex interactions with Electronic Blocks, the use of logic blocks and interacting block structures, which provide the greatest opportunities to learn about programming. Examples outlined throughout the paper, demonstrate how the simplicity of Electronic Block syntax allows for significant opportunities to explore programming concepts such as defining a problem, planning the solution, testing and debugging, and evaluating alternatives.

6.3. Evaluating the Electronic Blocks in Educational Settings

Theoretical evidence points to the Electronic Blocks being a powerful tool for young children to experience programming without the use of a computer. The next step in this project is to carry out evaluations in school settings to the test this hypothesis. The Electronic Blocks are currently being implemented following the design outlined in this paper. They will be taken into both preschool and primary school settings for complete evaluation.

7. References

[1] Adams, T. (1996). Logo environments: The evolution of the language. In J. Oakley (ed.), Logo in Australia: Selected readings (pp. 1-16). Richmond, Vic: Computing in Education Group of Victoria.

[2] Bredekamp, S., & Copple, C. (Eds.). (1997). Developmentally appropriate practice in early childhood

education. (Revised ed.). Washington, D.C.: National Association for the Education of Young Children.
[3] Cuffaro, H. K. (1984). Microcomputers in education: Why is earlier better? Teachers College Record, 85, 559-568.
[4] Green, T. R. G., & Petre, M. (1996). Usability analysis of visual programming environments: A 'cognitive dimensions' framework. Journal of Visual Languages and Computing, 7, 131-174.
[5] Gullo, D. F. (1992). Development and characteristics of kindergarten-age children. In L. R. Williams, & D. P. Fromberg (eds.), Encyclopaedia of early childhood education (pp 206-207). New York: Garland Publishing Inc.
[6] Kahn, K. (1996). ToonTalk™ – An animated programming environment for children. Journal of Visual Languages and Computing, 7, 197-217.
[7] Kay, A. (1994). Observations about children and computers. Advanced Technology Group, Learning Concepts Group, Apple Research Laboratory Research Note No. 31. [Online]. Available: http://www.atg.apple.com/technology/reports/RN31.html
[8] Oakley, J., & McDougall, A. (1997). Young children as programmers: Fantasy or flight. In A. McDougall, & C. Dowling (Eds.), Learning in Logo microworlds. Richmond, Vic: Computing in Education Group of Victoria.
[9] Papert, S. (1980). Mindstorms: Children, computers and powerful ideas. New York: Basic Books.
[10] Raizen, S. A., Sellwood, P., Todd, R. D., & Vickers, M. (1995). Technology education in the classroom: Understanding the designed world. San Francisco: Jossey-Bass Publishers.
[11] Resnick, L. (1987). Education and learning to think. Committee on Mathematics, Science, and Technology Education, Commission on Behavioural and Social Sciences and Education, National Research Council. Washington, D.C.: National Academy Press.
[12] Resnick, M., Bruckman, A., & Martin, F. (1996) Pianos not stereos: Creating computational construction kits. Interactions, 3 (5), 41-50.
[13] Sheingold, K. (1987). The microcomputer as a symbolic medium. In R. D. Pea, & K. Sheingold (Eds.), Mirrors of minds: Patterns of experience in educational computing (pp 198-208). Norwood, NJ: Ablex Publishing Corporation.
[14] Sloan, D. (1984). On raising critical questions about the computer in education. Teachers College Record, 85, 539-547.
[15] Smith, D. C., Cypher, A., & Schmucker, K. (1996). Making programming easier for children. Interactions, September – October, 59-67.