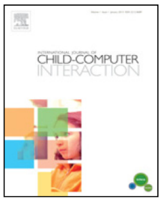
Contents lists available at ScienceDirect

International Journal of Child-Computer Interaction

journal homepage: www.elsevier.com/locate/ijcci



Designing for learning mathematics through programming: A case study of pupils engaging with place value

Laura Benton [a,*], Piers Saunders [a], Ivan Kalas [a,b], Celia Hoyles [a], Richard Noss [a]

[a] *UCL Knowledge Lab, UCL Institute of Education, University College London, 23-29 Emerald Street, London, UK*
[b] *Department of Informatics Education, Comenius University, Bratislava, Slovakia*

A R T I C L E   I N F O

Article history:
Received 7 February 2017
Received in revised form 28 September 2017
Accepted 20 December 2017
Available online xxxx

Keywords:
Children's programming
Computing education
Mathematics
Scratch
Broadcasting
Place value

A B S T R A C T

This paper focuses on a major part of a two-year intervention, ScratchMaths (SM), which seeks to exploit programming for the learning of mathematics. The SM hypothesis is that given the right design of curriculum, pedagogy and digital tools, pupils can engage with and express important mathematical ideas through computer programming. We describe the overall design of SM and as an illustration of the approach, we elaborate a more detailed description of the specific SM activities that seek to harness the programming concept of 'objects communicating with one another' for the exploration of the mathematical concept of *place value* through a syntonic approach to learning. We report a case study of how these activities were implemented in two primary classes. Our findings constitute a kind of existence theorem: that with carefully designed *and* sequenced learning activities and appropriate teacher support, this approach can allow pupils to engage with difficult mathematical ideas in new, meaningful and generalisable ways. We also point to the challenges which emerged through this process in ensuring pupils encounter these mathematical ideas.

1. Introduction

There was a body of research undertaken during the 1980s and 90s that explored the potential beneficial impact of learning to program on pupils' mathematical thinking [1]. Researchers designed mathematical activities that exploited the affordances of a specific programming environment to explore and express a range of mathematical ideas through the programming language. Many of these researchers were inspired by the work of Seymour Papert, and built microworlds in the Logo programming language [1]. Their research was guided by a constructionist approach to learning, which was viewed as a process of building knowledge structures that happens most effectively when the learner is actively engaged in "constructing a public entity" [2]. The approach endeavours to ensure that the learning activities embed "powerful ideas" [3] — ideas that are potent in their use (both epistemologically and personally), in their connections with other disciplines and their fit with a learner's personal intuitive knowledge [4].

Researchers also identified challenges experienced in learning to program, for example Lewis [5] and Resnick et al. [6] pointed to limitations of the programming environment, such as attention on mastering the programming syntax rather than the semantic meaning of the code as well as the specific skills/knowledge required by teachers to provide the necessary guidance or challenge to novice learners. New blocks-based programming languages, such as Scratch (derived from Logo), help to overcome difficulties in syntax, thus making programming more accessible to a wider range of learners and teachers [7]. These languages are designed to make some complex concepts more accessible, with visual cues such as colour, shape and constrained nesting to indicate usage, flow and scope [5,8].

The research reported here is based on the ScratchMaths (SM) project, which set out to exploit the resurgence of the teaching of programming (now commonly referred to as 'coding') in primary schools in England along with the programming functionalities for younger learners that have become available. Several curriculum designers have recognised the potential that Scratch offers the learning of mathematics, largely with a focus on geometry, but also exploring positive/negative numbers, building arithmetic and algebraic expressions, using the in-built coordinate system as well as developing mathematical thinking skills and positive attitudes towards mathematics [9–14]. SM aimed to make problematic parts of the mathematics curriculum personally meaningful, enabling more pupils to gain a deeper understanding of mathematical ideas through the programming activities used to express them. A central design focus was to exploit *syntonic learning opportunities*, that is to support learners in considering a problem from the

* Corresponding author.
E-mail addresses: l.benton@ucl.ac.uk (L. Benton), piers.saunders@ucl.ac.uk (P. Saunders), i.kalas@ucl.ac.uk, kalas@fmph.uniba.sk (I. Kalas), c.hoyles@ucl.ac.uk (C. Hoyles), r.noss@ucl.ac.uk (R. Noss).

https://doi.org/10.1016/j.ijcci.2017.12.004
2212-8689/© 2018 Elsevier B.V. All rights reserved.

Table 1
Ten tools for teaching for transfer (based on [26,27]) and examples of these techniques found within the existing research on teaching programming and mathematics.

Hugging techniques	Bridging techniques
1. *Setting expectations:* Indicating how to directly apply learning content in transfer context (e.g. identifying difficulties of fractions and discussing how they could be represented within Logo [28])	6. *Anticipating applications*: Discussing potential transfer contexts of learning context (e.g. setting a task to program a mathematics game in Scratch to facilitate number understanding of younger pupils [29])
2. *Matching*: Adjusting learning activity to reflect transfer context (e.g. identifying squares in environment and then using Logo programming to draw squares [3])	7. *Generalising concepts*: Identifying generalisable aspects (e.g. principles, rules, ideas) of learning content (e.g. sequencing questions to encourage generalisation of algebraic concepts encountered in Logo [30])
3. *Simulating*: Approximating learning content to transfer context through role play, acting out etc. (e.g. encouraging a pupil to "play Turtle" to imagine how to draw a square in Logo [3])	8. *Using analogies*: Representing the learning content through a different topic using an analogy
4. *Modelling*: Showing or demonstrating use of learning content within transfer context (e.g. providing examples of angles with rotation to support drawing of shapes within Logo microworld [31])	9. *Parallel problem-solving*: Exploring parallels and differences in applying learning content within two different problem contexts
5. *Problem-based learning*: Requiring the use of learning content to solve a problem within the transfer context (e.g. learning problem-solving skills through making games [32])	10. *Metacognitive reflection*: Promoting planning, monitoring and on their own thinking (e.g. design debriefings to encourage reflection on connections between game design choices and maths learning [14])

perspective of the programmable object, which "represents a 'resonance' between external forms and concepts and what people know about themselves" [3,15]. The concept of syntonicity has typically been used when exploring geometric ideas through programming [3]. We sought to extend its scope to other areas of mathematics and through a design research process aimed to identify concepts in the primary mathematics curriculum that we hypothesised would benefit from this approach.

The Scratch environment allows multiple objects (called sprites) to be programmed so that they act in parallel by programming interactions between them through a kind of message-passing, known as 'broadcasting'.[1] In this paper we explore how to exploit this programming functionality of objects communicating with one another as an alternative, meaningful and generalisable way for pupils to engage with the mathematical concept of place value — where the places 'interact', allowing pupils to play with the ideas directly rather than simply learning about them. We first present the background to the SM project and the overall design of the SM curriculum, before turning to the case study research that forms the empirical core of the paper.

2. Background

Learning something in one context, such as programming, and subsequently utilising this skill or knowledge in another context is commonly referred to as 'transfer' (see for example [16,17]). The traditional view of transfer is that learning x brings about an ability to understand y where, in some real sense, x and y are epistemologically different, an approach that has been subjected to critical scrutiny (e.g. [18]).

The early Logo experiments constituted a widespread attempt to explore the impact of computer programming on children's mathematical understanding [1,19]. Clements and Sarama [20] propose that programming in Logo could "serve as a transitional device between concrete experiences and abstract mathematics". However, there were conflicting findings, which may in part be explained by the different approaches to and reasons for teaching programming, as well as to the differing levels of facilitation of the connection-making process between the different domains, usually offered by a teacher [see 21, Ch. 7]. In fact Delclos et al. [22] state that "no content, standing alone, can spontaneously produce generalisable learning" and highlight the important role of the teacher in mediating the learning through programming languages and supporting transfer.

Some researchers have identified teaching techniques that facilitate transfer, such as *hugging* and *bridging*, distinguished by Salomon and Perkins [23]. Hugging seeks to make the teaching in the new domain as similar as possible to the original context [23–25]. Bridging by contrast promotes facilitating abstraction and connection-making processes by, for example, making analogies or teaching problem-solving strategies. Salomon and Perkins [23] suggest that using a combination of these techniques maximises potential transfer opportunities. Table 1 summarises a set of hugging and bridging techniques, based on the mechanisms for transfer proposed by Salomon and Perkins, and expanded by Fogarty et al. [26]. It lists 'ten tools for teaching for transfer' adapted specifically for mathematics by Jones et al. [27], and how these techniques have been used within programming and mathematics.

We do not see mathematical learning as a spin-off from learning to program, therefore we have considered whether children taught to program in Scratch can express some of their mathematical thinking in programming with the support of carefully designed materials *and* teachers who have been 'trained' in how to *use* Scratch to *learn* mathematics. Within mathematics there is often a disconnect between concept and procedure, with a dominant focus on the latter, despite overwhelming consensus that what is needed is a judicious mix of the two [33]. The difficulty for the pupils is to 'keep hold' of what the symbols *mean,* and simultaneously to let go of the meaning when manipulation of the symbols *is* a realistic objective.[2] A pervasive example of this is in the way children are introduced to the algorithms of arithmetic, which is the focus of our investigation here.

3. ScratchMaths curriculum design

In the SM project a two-year curriculum was designed for the 9–11 primary age group (Years 5 and 6), aligned to the English National Computing and Mathematics Primary Curriculums [34], and promoting the teaching of carefully selected core ideas of computer programming alongside specific mathematical concepts. The content was divided into six modules, three modules to be taught per year (see Table 2). The project team followed a design research process to develop the curriculum content and pedagogic approach (termed the 5Es framework[3]), trialling the activities in a number of 'design schools' and iteratively redesigning based on the outcomes of these trials (for further details see [35,36]).

[1] The key attributes of which are adopted from more general mechanism of event handling (in particular the mechanics of user-defined event handling).

[2] For example, a Logo program such as REPEAT 3 [FD 50 RT 120] contains within itself some information about the number of symmetries, and an application of a very powerful theorem (the Total Turtle Trip theorem). In short, the symbols bring alive aspects of the *structure* of an equilateral triangle.

[3] Which includes the constructs Explore, Explain, Exchange, Envisage and bridgE.