Evaluating Scratch to introduce younger schoolchildren to programming

Amanda Wilson and David C. Moffat

School of Engineering and Computing,
Glasgow Caledonian University,
Glasgow, Scotland, UK
D.C.Moffat@gcu.ac.uk

Abstract. The Scratch system was designed to enable computing novices, without much programming experience, to develop their creativity, make multimedia products, and share them with their friends and on a social media website.
It can also be used to introduce programming to novices. In this initial study, we used Scratch to teach some elementary programming to young children (eight years old) in their ICT class, for eight lessons in all. Data were recorded to measure any cognitive progress of the pupils, and any affective impact that the lessons had on them.
The children were soon able to write elementary programs, and moreover evidently had a lot of fun doing so. An interview with their teacher showed that some of the pupils did surprisingly well, beyond all expectations. While the cognitive progress is moderate, the main advantage to Scratch in this study seems to be that its enjoyability makes learning how to program a positive experience, contrary to the frustration and anxiety that so often seems to characterise the usual learning experience.

Keywords: POP-I.A. learning to program; POP-I.B. choice of methodology; POP-II.A. novices, schoolchildren; POP-III.B. smalltalk; POP-III.C. visual languages; POP-III.D. visualisation; POP-IV.A. exploratory; POP-V.B. case study; POP-VI.E.

1 Introduction

Computing technology is increasingly important in the modern world, which could not function without it. One might expect greater numbers of students to want to learn about computing; but numbers of students at school and university are falling in the industrialised world.

The situation in the UK, for instance, is approaching crisis point, as recently documented by the UK's Computing Research Committee. According to their report (UKCRC, 2010), the numbers of school pupils taking Computing or ICT (Information and Communication Technologies) courses has "collapsed" by about a third in less than five years; and the consequences for university intake have been severe.

One of the major problems identified is that Computing in schools is typically confused with ICT, and pupils are taught basic skills in office applications like word-processing and spreadsheets. Their teachers themselves often have no formal education in computing, and cannot communicate enthusiasm or understanding about what happens inside a computer to make it work. In particular, there is little introduction to programming in some schools, and what there is can easily lead to intimidation of the pupils rather than enlightenment. As a result, they may leave school feeling that programming is mysterious and difficult, or frustrating and boring. It is no wonder then, if they choose not to pursue computing at university and in the workplace.

The problem may be tackled by making introductory programming both easier and more fun, and there are several attempts to achieve this. The Scratch[1] system from MIT (Resnick et

[1] Home website: http://www.scratch.mit.edu/

al., 2009) is a simplified visual programming system in which it is relatively easy to manipulate multimedia objects, without much preparation. It is a leading candidate to help introduce children to programming, and the subject of the present study.

Scratch has been used by some enthusiastic teachers in schools in the USA and the UK for extra-curricula activities (like after-school clubs), and anecdotally they are pleased with the experience. It has been used for introductory programming at some universities, which have gone so far as to publish evaluations of it; but there is little evaluation to date of its use with the intended age group of middle school pupils.

In the present study, we made an initial evaluation of Scratch for school pupils, where we deployed it in their IT lessons for eight weeks.

1.1 What is Scratch?

Originally inspired by Papert's work (Papert, 1980), Scratch was intended by Resnick to support creative work with multimedia (Maloney, Peppler, Kafai, Resnick, & Rusk, 2008) in "computer clubhouses" or after-school learning centres for children from deprived communities, and was first deployed in 2005. Those children enjoyed multimedia "mash-ups," like the sampling techniques used in the pop music they liked, which is why the new system came to be called "Scratch."

The focus of Scratch is on making multimedia products, and sharing them in the large and active online community hosted by the project website. This is intended to enable and develop children's creativity, but also to introduce them to programming, in a fun way.

Visual programming The way programs are written in Scratch is by fitting "blocks," together rather like toy Lego bricks; or pieces of a jigsaw puzzle. In this respect, the programming language in Scratch is a "visual language", (Green & Petre, 1996).

The blocks can only fit in ways that make sense, because of their shapes, so it is not possible to get error messages from the compiler. This is a great relief for introductory programming, and saves the learner from much of the heartache traditionally forced on them by textual languages. Learners in Scratch are not bullied by the compiler when they forget a semicolon or have mismatched brackets, because such errors are not possible. To the extent that novices get frustrated or daunted by floods of compiler errors, the visual language in Scratch gives it strong appeal for educational purposes.

Fig. 1. Screenshot of a classic "Hello World!" program in Scratch

A traditional first program is shown in Fig. 1, where the default character (on the right, in the canvas window) responds when you click on the sprite, with a thought-bubble that appears for two seconds, and then says "Hello World!" in a speech bubble. The script that does this is shown in the bottom middle window, which was made by drag-and-drop from the palette of blocks in the left window.

Other palettes are available in the upper left window, which include blocks for program control logic (selected), blocks to move sprites around the canvas, blocks to draw on the canvas, blocks to sense events like collision detection, blocks for playing sounds, and so on. There are blocks for arithmetic, boolean and string operations, and for variables too.

1.2 Related work — evaluation of Scratch for novice programmers

The Scratch system has been a big hit with its intended users, in computer clubhouse environments, as reported by the developers (Maloney et al., 2008). The children spent more time working on Scratch than with any other package they had available to them. It seems clear that Scratch succeeds very well in fostering creativity and in social sharing of the multimedia products.

It was envisaged from the outset that while this project was to introduce computers to deprived areas, the educational benefits would be researched at a later date (Resnick, Kafai, & Maeda, 2003). Because Scratch is a new system, there have only been a few studies of its use in teaching programming, so far.

In one study, Scratch was used at Harvard university (Malan & Leitner, 2007) , where it was used to introduce novices to programming before their transition to Java. There was an almost total approval of Scratch amongst the learners who were true novices; and the only learners who disagreed that it was useful to them were the few people who had already some experience of programming.

Another pilot study was in the USA with 8th grade girls at middle school, where the aim was to see whether the pupils would learn to appreciate the basics of programming in the span of a three-hour workshop (Sivilotti & Laugel, 2008). The girls were not complete novices, all having used either Scratch or Lego Mindstorms or Logo before. They reported feeling that they had learned something worthwhile and how much fun they had had (average 3 and 3.4 respectively, on a 4-point scale).

2 Method: to try Scratch out in a real classroom at primary school

The existing studies above have evaluated Scratch either informally, in after-school activities, or more formally with older or more experienced students. It was generally observed that Scratch was fun to use, and there were some observations about learning ocurring.

The purpose of the present study is to evaluate the use of Scratch in school lessons as an introduction to programming for total novices, in a younger age-group at primary school. It focuses on two possible kinds of benefit: cognitive and affective; we are interested to know whether Scratch teaches concepts well, and whether it is fun to use for the younger age-group in a school context.

2.1 The school and pupils

The primary school chosen for this study is in a relatively deprived area of Glasgow, in Scotland. The class has twenty-one pupils, who are all eight or nine years old. One of us (AW)

approached the school teacher to offer taking her IT lessons for the whole term of eight weeks. The teacher agreed, being interested to see how well the lessons would go with Scratch, as compared to the normal ICT lessons that she gave the pupils, in which they would typically use office applications or surf the web.

2.2 The lesson plans

Scotland is currently renewing the schools curriculum, and so the lesson plans were drawn up with the new *Curriculum for Excellence* (LTS, 2010) in mind. That way, the teacher can continue to use the lesson plans later on, if she so chooses, because the lessons satisfy the desiderata of the new curriculum.

For each of the eight weeks, there was a one-hour lesson. At the start of each lesson, we showed all the pupils what they would do next, on a whiteboard at the front of the class. Then they went to work in pairs at the computers, to try and achieve the task using Scratch.

The tasks were to make a sprite move around the canvas, either to make patterns, or to visit certain locations in turn and end up at a target location. Each week's lesson was a little more complicated than the previous one.

In order to illustrate to the children what task to achieve for each lesson, the demonstration was given either on the whiteboard, or using a small remote-control toy, which was a toy robot. Some children would call out instructions to the one with the remote control, who would then control the toy robot. By this kind of concrete programming (Demo, Marciano, & Siega, 2008) the children can think through what sequence of actions is required to get the robot to its destination, and they are then ready to try the task with Scratch.

First visit – set baseline of understanding Find out what the children might already know about programming.

Illustrate the concept of algorithm with an example of making breakfast: (1) Get cereal box, (2) get bowl, (3) get milk, (4) pour cereal into bowl, (5) pour milk into bowl, etc. Emphasize the importance of getting the order right (to help understand sequencing later on).

Lesson one – introduction to Scratch Introduce the children to Scratch, with a worksheet that shows a couple of program "blocks": (a) to "move 10 steps" and (b) to "turn right 90 degrees". The children can experiment with the effects of these blocks on the cat character, and they can try different numbers of steps to move or degrees to turn. Then they can look at the other blocks available in the palette and come up with their own ideas to try out for the rest of the lesson.

Lesson two – introduce "sequence" Hold a class discussion about how a program can make shapes on the canvas, by using the pen (as with turtle graphics). Then demonstrate the program with a remote control toy, and let the children go to the computers to put the programs into Scratch and try them out.

Lesson three – first class test The exercise is to write a program to move the sprite (cartoon character) across the canvas, while on the way passing over each of the coloured shapes that have previously been drawn on it by the tutor. This cannot be done with a single straight line, so the children have to make a route out of straight segments joined by 90 degree turns (see Fig. 2). Before trying to do this in Scratch, they first work out the path they want and the instructions required to draw it, and they write their little program on the paper worksheet.

Their worksheets are collected for later marking to monitor the pupils' progress. Marks for this test (out of eleven) are awarded according to how much of the path the child manages to produce:

- did he or she draw the line from start to finish?
- did he write down correct instructions, that get the sprite to the end?
- did he put in the correct turns, in direction and in degrees?

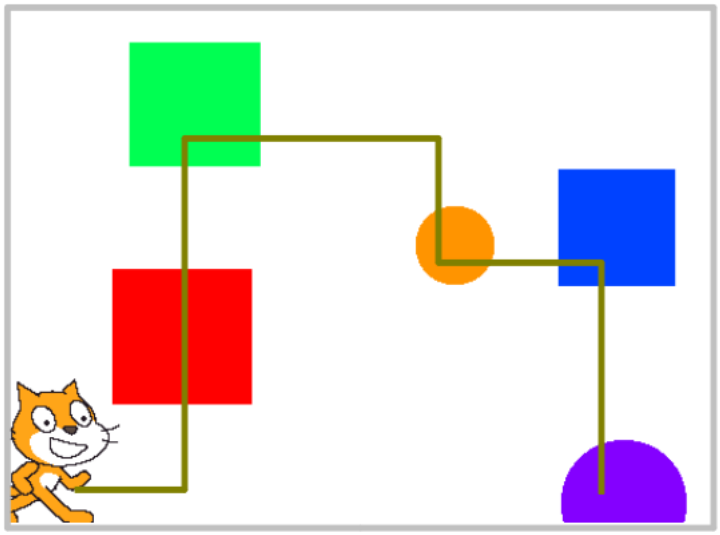

Fig. 2. Lesson-3 exercise, showing a route that visits all the shapes

Lesson four – iteration Introduce the children to the "repeat" block, as a way to make repetitive scripts shorter. Show an example script of a line segment followed by a quarter turn, and then enclose it inside a repeat block (see Fig. 3), that runs it four times… to make a square.

Fig. 3. Lesson-4, showing a repeat-block for iteration

Lesson five – selection Introduce the class to conditionals, by using *if- else*-statements to make their sprite rebound when it collides with the endge of the canvas, or another sprite.

Lesson six – coordination and synchronisation Using the "broadcast" block, which sends messages to any other blocks that care to listen, the children can make a short animated sequence in which two sprites talk to each other (see Fig. 4).

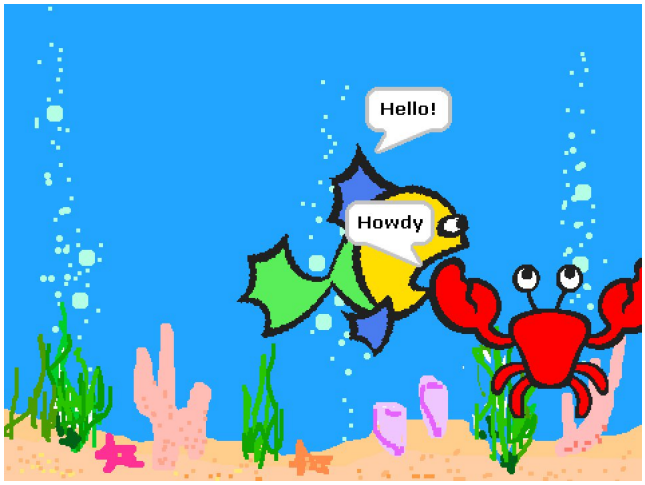

Fig. 4. Lesson-6, showing an animation with sprites conversing

Lesson seven – the Scratch cards A set of twelve cards is available to download and from the Scratch website, each of which helps the learner to explore another feature of Scratch. One card shows how to play sounds, for example, and another one shows how to make a sprite follow the mouse cursor.

Show the children how the cards suggest things to try in Scratch, and let them work their way through the set.

Lesson eight – second class test The exercise in this lesson is to be marked afterwards, and includes several tasks, of which one is comparable to the first class test from Lesson-3. The latter task is similar to the one from the first class test, except that it has a set of shapes in different locations, necessitating a different path to negotiate them.

Unfortunately, because the exercises in this lesson were longer, the children did not all finish them, and some rushed their answers. For this reason, we did not use the results to compare with Lesson-3.

Week nine – after the lessons, a final class test We set the class another test (Test-3), similar to the first class test in Lesson-3, and this time without extra time-pressure. The results from this test were used for further analysis (see below).

2.3 The measurements taken

We wanted to know how the lessons compared with the class's other, normal lessons in ICT. The two major factors were *cognitive* (how effectively they learned) and *affective* (how enjoyable the experience was, and how motivated by it the pupils were).

As well as some simple questionnaires for the pupils, their behaviour was observed during the lessons, and the teacher was interviewed for her reactions and opinions, as she knows the pupils well.

Cognitive measures In order to measure learning progress, the pupils were set some questions at two points during the term: the middle and the end (lessons 3 and 8). The questions were inspired by the Cambridge "ICT starters" syllabus for assessment of early progress in ICT skills (University of Cambridge International Examinations (CIE), 2010).

While the "ICT Starters" curriculum covers ICT skills from word processing and spreadsheets to email and web-browsing and authoring, it also includes *control*, which is more closely related to programming concepts. Children are to demonstrate control by giving simple commands to a device; and by using a sequence of commands to control a device, including inputs and outputs. The programming language to use for these activities is *Logo*.

Our tests and scoring schemes were based on their assessment ideas, which allowed the childrens' work to be judged and quantified as to the level of skill demonstrated. In order to show that children have developed some facility for control of a device, the curriculum requires that they produce a sequence of instructions that involve at least a certain number of line segments, and a certain number of 90-degree turns. The class tests were devised to embed these requirements into the tasks set for the children, in making a sprite navigate around the canvas, visiting various locations on the way.

Affective measures In order to measure how enjoyable the children found their lessons with Scratch, or whether they were growing at all frustrated, they filled in a brief log-sheet after each lesson, to say what they did in the lesson and how they felt about it. Rather than ask such young children to describe their feelings, the log-sheet had three cartoon faces (sad / neutral / smiling) which they could mark with a cross (see Fig. 5).

Fig. 5. Affect measure: a log-sheet that young children can easily understand

3 Results

There were twenty-one children in the class (5 girls and 16 boys), but some did not attend all the lessons. All were eight years old, except for the four nine-year-olds.

Nineteen pupils had a computer at home, but none of them knew what a computer program was before the lessons. They had never done any form of programming before, neither at school

with a teacher, nor at home. The pupils mostly thought that a program was something to do with the internet, or with computer games.

3.1 Cognition and learning

The tasks that we used as tests, and marked, were given in weeks three, eight and nine. Unfortunately the scores for the second test were low, because the children ran short of time in that class, so we use the scores from Test-3 instead. There was a problem with that test as well, in that it took place near the holidays, and several children were absent that day for that reason, and because of an infection that was going through the school at the time.

Leaving out the missing values, there were only $N = 12$ pupils that had scores for both Test-1 and Test-3. The mean score for the tests were 52% and 64%, respectively. Although this shows an improvement in the pupils' performance, the difference is not statistically significant at the 95% level (paired t-test, t = -1.741, df = 21.202, p = 0.09617).

3.2 Affective experience of pupils

At the end of each lesson, the children marked on their log-sheets how they felt about the lesson with Scratch. Answers were on a 3-point scale, shown by three cartoon faces which were either sad or neutral or smiling. The result averages are shown in Table 1, where the missing values for the lesson in week-2 are shown as blanks: there was no time to fill in the log-sheets that week.

Table 1. How the pupils felt about their lessons (sad, neutral or happy)

lesson :	1	2	3	4	5	6	7	8
happy	19	-	20	20	18	18	18	15
neutral	0	-	0	0	2	1	2	3
sad	0	-	0	0	0	0	0	0
absent	2	-	1	1	1	2	1	3

It is clear from these results that all the pupils enjoyed the lessons hugely. Nobody was ever sad, a few were neutral for at most two of the weeks, and all other marks were for smiley faces. In fact only five of the twenty-one pupils ever marked a lesson down to neutral.

3.3 Teacher's views

All the lessons were lead by one of us (AW), while the teacher watched and helped, because this was new to her as well as to the class. At the end of the term, she was interviewed for her personal assessment of her pupils' progress, because she knew them well and could compare their performance and enjoyment in our lessons with the way they were in other classes. Except for people's names, her answers are transcribed here *verbatim*, as follows:

Question: What expectations did you have at the beginning, and have they been met?
Answer: Without a doubt – they have been exceeded!

Question: How do you feel your class performed in the Scratch lessons compared to how they would perform in normal ICT lessons?

Answer: Far more enthusiastically. They picked it up very quickly and easily; I don't know why. Maybe it was the Scratch system, or maybe it was your tutoring. But they were more enthusiastic than in their other subjects. They were very keen, kept at the task, and didn't have to be told to keep quiet.

Question: How were they compared to how they would perform in maths lessons?
Answer: Much better.

Question: From at the test results, did any children do better or worse than you'd expected?
Answer: Yes:

- one did much better, as he seems to be really good on the PC. He is good at maths too, but finds language difficult.
- Another one made a great improvement.
- But I was surprised that two others did much worse than average in this class, while they are in the top group.
- *It's obvious to me the less able pupils are doing better with Scratch.* I'm surprised at those not doing well academically doing really well with Scratch. Some of them have language barriers as well.

Question: Did you enjoy the lessons?
Answer: Yes I did.

Question: Would you recommend Scratch to your colleagues as a tool to teach computing? Will you use Scratch yourself in future?
Answer:: Yes, without a doubt — it's a great tool for teaching. We'd like you to come again and show us teachers more about it.

4 Discussion and conclusion

It is clearer from the teacher's answers than from the other data just how much better the progress and behaviour was in these Scratch lessons, compared to other classes. Consider in turn the affective and cognitive factors at play.

4.1 Was affect important?

The level of enjoyment was consistently high, for all pupils and for every week. It was noticeable that the pupils were laughing quite a lot, and showing their work to each other, and to the teacher. Some might think this emotional side to be unimportant, or much less important that the cognitive effects such as evidence of learning; but we do not. Affect is important for learning, and not just as an accompaniment. Learning will hardly progress without motivation, and that is stirred and maintained by positive affect. The teacher of these children also seems to prize the fun that she sees in the class when they learn to use Scratch.

The teacher's remarks about some of the less able pupils doing very well were not entirely suprising to us – we had suspected that might happen for one or two children who were otherwise difficult to reach. But it certainly was a surprise to us that a couple of the academically strongest children did conversely: rather poorly. While they were normally in the top group, in our lessons their performance was amongst the lowest in the class. We hope to discuss this matter again with the teacher, and until then we cannot explain it.

There is little doubt that Scratch, in combination with the lesson plans we used, was a big hit with the whole class; all the pupils, and the teacher too. Our formal measure of affective reactions clearly showed that the lessons were enjoyable for all the pupils.

Was it merely about novelty? It is of course possible that this effect is entirely due to novelty, as the pupils had never seen Scratch before, nor their new tutor. That issue can only be entirely settled with a longer study. However, we note that eight weeks is quite a long time for a pure novelty effect to sustain, without the slightest fall. Note also that the "new tutor" had the pupils' attention for the first 5-10 minutes of each class, and after that they went to work on the computers in pairs, as usual. Perhaps it might be argued that the lesson plans introduced more novelty each week, with new facets of Scratch programming to explore, and that it was the continual recycling of novelty that sustained the novelty effect. By that argument, however, all forms of learning would be self-reinforcing by means of novelty cycling alone. The argument is thus a facile one, and we reject it in favour of concluding that Scratch has shown itself to be beneficial (and fun) for learning programming, and for more reasons than mere novelty.

4.2 Is there good news on the cognitive front?

Our formal measure of cognitive progress did show some improvement, but not enough to be statistically significant. The results were hampered by missing values; and perhaps it was too much to expect in only eight hours of lessons, albeit spread over eight weeks.

It may not have been the best idea to use our Test-1 and Test-3 in order to measure progress, because both tests were about the same subset of skills. That makes for a convenient comparison, but on the other hand it also misses any other learning that may take place. For example, in Lessons four and five the children learned about iteration and selection statements; but there is no need for them in the class tests, so any learning there would be missed by them. The only way that the tests would show learning is if the children improve their performance on the basic skills by practicing with Scratch while learning the more adanced ones.

In future work it would be an idea to plan out a more open-ended set of challenges, that would allow pupils to use things like iteration and selection, if they knew them; but would let them achieve success in a simpler, more tedious way, if they didn't know them yet. For example, a program with an iterative loop may be equivalent to a longer one where the loop has been unfolded. A sensitive marking scheme would be needed, to reward any flashes of inspiration and novel attempts that don't happen to work, but are still evidence of insight. Until then, we must conclude that the advantage of the Scratch system, now applied to teaching programming, has not been formally demonstrated in this study. Quantitative results did not significantly support it.

In order to prove that Scratch is better than any usual alternative, a controlled study where the alternatives are played off against each other would be required. Without doing that, however, we can still make a case in favour of Scratch. **First:** note that in only eight weeks the young children tackled the key elements of programming (sequence, iteration and selection); and more, besides. They were also introduced to synchronisation between scripts, and elementary multimedia effects. **Second:** imagine trying to achieve the same progress in the same time with a standard alternative – say with Java and Swing. Give the eight-year olds a nice IDE as well if you like, to help them along. **Third:** no, there is no third step, since the second step is already incredible, isn't it? The present authors certainly cannot imagine it happening, unless perhaps with child geniuses.

Other curious observations It was good to see that the Scratch lessons were able to reach some of the pupils who are known to have difficulty with other classes; with "language" in particular.

However, it is puzzling that some of the ordinarily stronger pupils did uncharacteristically poorly in the Scratch lessons. This issue is a matter for further investigation.

For the time being, it seems safe to conclude that Scratch looks like a rather successful way to introduce programming concepts to children as young as eight years old; even including some who suffer from a degree of learning difficulty.

It remains to be seen whether the pupils will maintain their enthusiasm about programming when they later encounter more conventional languages, with fussy syntax; but at this point we are encouraged at the prospect. We are thinking of ways to use Scratch in future as part of our outreach programmes, for example to help schools learn how to deploy it in the classroom.

4.3 Ideas that may help explain success

Future work could look at the features of Scratch, and attempt to break down which ones are the most crucial in achieving the generally happy results. The two major features of Scratch are the visual language, and the multimedia environment. Each one may well have a double benefit, as we shall now speculate.

Benefits of visual language Because the programming language is visual, it benefits both the cognitive and affective sides of learning. The essence of programming includes the key concepts of sequence, iteration and so on; matters of syntax in a textual language are relatively superficial, and so are less important. In order to learn the key concepts, however, the pupil has to first master enough syntax to support them. Therefore, the material has to be learned in the wrong order, with the lower priority syntax taking precedence. Using a visual language does not solve the problem of syntax, since the student will have to tackle that later when learning a second, more traditional language; but it does postpone the problem until the student has grasped the fundamentals. In this way, the memory load is kept manageable at all times, instead of overloading the mind all at once with barely sensible detail. This is a big *cognitive* benefit.

The *affective* benefit of learning a visual language is simply that a lot of heartache is avoided, which is to say that the potential for negative affect is neutralised. It may well be more important to prevent negative emotions from intruding into learning, than to encourage positive ones. Therefore, this benefit is particularly strong.

Benefits of multimedia platform The *affective* benefit of the multimedia environment is fairly obvious: it's fun to play with. Consequences of that are that the pupil will happily spend more time with the system, including leisure time, and will also tend to explore the system more, and try out new program blocks and other things that a more sober system would not encourage. Such exploration, naturally, should accelerate learning.

The chief *cognitive* benefit of the multimedia elements, that are fairly easy to manipulate, may be the instant and vivid feedback that makes the internal workings of the programmed system so much more apparent to the learner. This is because the program statements can be so closely related to the multimedia elements that the sprites and sounds and events in the 2D virtual world are (virtually) the program itself in motion.

A more traditional program for a novice task would involve manipulation of data structures, some calculations, and eventual printouts of results; but to make the internal workings of the

algorithms more visible, suitable print statements would need to be set in the program, which can be a tedious process that also requires some astute thinking on the part of the poor novice to place them optimally. Yet again, the learner could become overwhelmed; either that or could become less ambitious, and rather reconciled instead to slower, more tedious progress, soon to be followed by boredom.

It is noteworthy that these affective and cognitive benefits are not independent; rather, they tend to feed back into each other. This reinforces our view that the affective side of programming is in its own way at least as important as the cognitive side. If that is so, then an ideal educational system to help learn how to program should be designed with as much attention paid to the learner's emotional state as to the cognitive dimension.

How many such systems or approaches are out there? We know of at least one.

5 Acknowledgements

Thank you to the teachers at the school and their pupils who took part in this study; and many thanks also to the kind and careful reviewers.

References

Demo, G. B., Marciano, G., & Siega, S. (2008). Concrete programming: Using small robots in primary schools. In *Proceedings of 8th IEEE international conference on advanced learning technologies* (p. 301-302). IEEE computer soc.

Green, T. R. G., & Petre, M.(1996). Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework. *Journal of Visual Languages & Computing*, *7*(2), 131–174.

LTS. (2010). *Curriculum for Excellence. Technologies: experiences and outcomes* (Govt. Rep.). Learning and Teaching Scotland (LTS). (Available at website: http://www.ltscotland.org.uk/curriculumforexcellence/technologies/)

Malan, D., & Leitner, H. (2007). Scratch for budding computer scientists. In *38th sigcse technical symposium on computer science education* (Vol. 391, pp. 223–227).

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N.(2008). Programming by choice: Urban youth learning programming with Scratch. *Proceedings of the 39th SIGCSE technical symposium on Computer science education – SIGCSE '08*, 367.

Papert, S.(1980). *Mindstorms: Children, computers, and powerful ideas* (2 ed.). Basic Books.

Resnick, M., Kafai, Y., & Maeda, J.(2003). *A networked, media-rich programming environment to enhance technological fluency at after-school centers in economically-disadvantaged communities.* (Proposal to the National Science Foundation, USA; project funded 2003-2007)

Resnick, M., Silverman, B., Kafai, Y., Maloney, J., Monroy-Hernández, A., Rusk, N., et al. (2009, November). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60–68.

Sivilotti, P. a. G., & Laugel, S. a. (2008, February). Scratching the surface of advanced topics in software engineering. *ACM SIGCSE Bulletin*, *40*(1), 291.

UKCRC. (2010, January). *Computing at School : the state of the nation.* Available at website: http://www.ukcrc.org.uk/.

University of Cambridge International Examinations (CIE). (2010).