Learning graphical programming: An evaluation of KidSim™.

David J Gilmore, Karen Pheasey, Jean Underwood & Geoffrey Underwood

ESRC Centre for Research in Development, Instruction and Training
Psychology Dept
University of Nottingham
Nottingham, NG7 2RD, UK
dg@psyc.nott.ac.uk

KEYWORDS: Programming, Graphical programming, abstraction, educational technology, learning.

ABSTRACT: This paper presents part of an evaluation of a new children's programming environment, developed by Apple Computer Inc. for 10-13 year old children. We studied 56 children, generally working in groups of 2-3, using KidSim™ for between 2-12 hours, over a period of between 2 days and 3 weeks. The results show that children of this age can readily learn to master the programming environment, and that they greatly enjoy using the system – indeed in most cases it clearly fired their imaginations. However, questions remain about the level of programming abstractions that they were able to understand.

The evaluations have led, however, to a small set of changes in the KidSim™ environment, all of which are intended to support improved comprehension of these abstractions. Further evaluations will be needed to discover these changes can maintain the motivational advantages of the present system, and yet improve the system's educational impact.

INTRODUCTION TO KidSim™[*]

Smith (1993) describes the programming environment KidSim™, developed by Apple Computer Inc. as an end-user programming environment targeted at children aged between 10-13. KidSim™ is a wholly graphical programming environment, containing agents for whom the children can construct graphical production rules which will move them around a 2-dimensional world.

Many of the ideas embodied in the system are derived from teachers' suggestions following the use of HyperCard for modelling Dewdney's 2-d world "The Planiverse" (Dewdney, 1982).

KidSim™ is still in the process of being developed and the version we used for these evaluations was an early prototype. It was extremely similar to that described in Smith (1993). Being a prototype it consumed large amounts of disk space and RAM and this imposed constraints on the evaluations in ways described below. By contrast, production versions of KidSim™ are expected to run on home machines, with minimal RAM requirements.

All interactions, whether programming or drawing (of the 2-dimensional world) are by direct manipulation. Except for naming agents, and maybe naming some of their characteristics, the children have no need to use a keyboard. The appearance of the world, objects within the world and of the agents themselves is all under the control of the children, through simple drawing tools.

The basic programming architecture is of a graphical production rules in which the interpreter tries to match the world around an agent to the left-hand side of one of their rules. When such a match is found then the world around the agent is changed to that represented by the rule's right-hand side. Each agent can have multiple rules and a world can contain multiple agents.

The 'world-around-an-agent' is of flexible size, though the size must be same for the left- and right-hand sides. It could be simply an agent and one adjacent space (the world is divided into square spaces), or it could be a large rectangle surrounding

[*]KidSim™ is a registered trademark of Apple Computer Inc.

the agent by any number of spaces in differing directions. The whole of the world defined by the rule has to match in order for the rule to be triggered.

PROGRAMMING ISSUES

At CHI'92 a special workshop on end-user programming was held (see Gray et al. 1993). at which there was lengthy discussion about both the potential and the interfaces for end-user programming. Rather pessimistically Gilmore presented an argument that people were assuming that end-users could acquire a complete conceptual grasp of programming. including the tough concepts of abstraction. modularity and analysis. Arguing that abstraction was the most important concept, it was claimed that end-users would not really acquire programming skills unless the interface provided concrete embodiments (visualisations) of abstraction.

As an example one can consider the abstract concept of a variable in HyperCard. Many Hypercard programmers do not use variables. since they are uncertain of their exact status and usage. However. people do use fields quite freely, and then they adapt to the slightly abstract notion of invisible fields (occasionally made visible when debugging). In fact. apart from long-term storage and speed. there are no differences between variables and invisible fields. Hypercard provides an ideal vehicle for scaffolding the concept of variable out of domain-specific text fields. What is not clear is whether the variable concept acquired is genuinely abstract. or whether it is HyperCard specific.

Unfortunately. this is not a readily testable idea. since HyperCard is neither a novice. nor an end-user programming environment. The discussion at CHI'92 was mixed – some believed that end-users would not be able to (or should not have to) learn any abstractions. whilst others felt that a good scaffolding environment could lead them through to general programming skills.

To date there have been no tests of end-user programming skills and knowledge – the studies have focused on task analysis and documenting the actual activities undertaken. However, there is an interesting and striking parallel here with the use of computing in classrooms. Indeed. maybe children are the ultimate end-user programmers.

Classroom computing

Papert (1970) argued that the study of programming is intellectually beneficial and for a number of years the notion persisted that programming was a "new Latin" which would promote good. domain-independent thinking skills in our children. Initial studies with LOGO offered the possibility that these claims might actually be true. but then more detailed and more rigorous studies came along which were generally unable to find any evidence of transfer from programming to general problem-solving skills (e.g. Mayer. Dyck & Vilberg, 1986).

However. most of these studies had a very narrow view of transfer. and used languages (usually LOGO) which embodied a narrow view of programming. And one of the key problems in the studies was that too many children acquired too little expertise in programming (e.g. Kurland, Pea. Clement & Mawby, 1986).

One of the key factors about programming skills is that most key concepts seem to be acquired when people are trying to solve their own problems. rather than the exercises provided by the teacher. Programming seems to be inherently a 'discovery learning' domain. Many of the early LOGO and other studies used fairly traditional instructional regimes. in which programming was taught through language features. code templates. procedural skills (e.g. planning and debugging) and finally general problem-solving. It isn't perhaps too surprising that children showed little transfer of this knowledge.

Linn & Dalbey (1985) showed how quality of instruction was of prime importance in determining children's programming success. with 'exemplary instruction' (which emphasised design skills and general transferable skills) advancing the students furthest along the chain of cognitive accomplishments. However. even here the only example of a general transferable skill was an understanding of a general sorting algorithm.

KidSim™. therefore. offers a chance to look at classroom computing in a manner which goes beyond most of the studies conducted so far. It offers children the possibility of studying problems of their own choosing, in a context which does not strongly associate programming with Maths or Science. or any other specific discipline. And yet. KidSim™ contains tough, general programming abstractions to be understood (e.g. the concept of abstraction itself. variables. a black box system).

At the current stage of development. our primary goal was to provide an early formative evaluation. hoping that our evaluations of KidSim™ can inform not only classroom computing, but the development of end-user programming systems too.

THE EVALUATION

If children can learn about abstraction and modularity. through good interface and environment design (rather than by instruction), then the potential for truly powerful end-user programming is more promising.

KidSim™ provides an opportunity to investigate this question. The nature of the prototype at the time of this evaluation constrained us to using machines in our Department, rather than being able to take KidSim™ into schools. It should also be pointed out that there was practically no documentation available about the system for the children to use, and the two research staff who were always present had themselves only been using KidSim™ for 2-3 weeks prior to the evaluation.

These constraints meant that we were engaging in a very conservative evaluation, since the system was both slow and unreliable, both seemingly undesirable properties in software intended for 11 year old children. Because of these constraints we decided against engaging in a serious educational evaluation, preferring instead to concentrate on interface issues and the children's general understanding of programming in KidSim™.

Evaluators
In total we studied 56 children for varying lengths of time. The majority of these came as a class from a local school for 3 afternoons (two classes were used, one aged 11/12 and the other 13/14). Other children in the study were the 11-12 year old children and friends of our colleagues. These latter children tended to come in for two or three whole days.

In most cases the children worked in groups of 2-3, though sometimes groups coalesced into larger groups. Also, sometimes some of the children preferred to work alone.

The data reported here relate to the 12- and 14- year old children from the local school. This was a total of approximately 32 children, who worked in groups of 2-3 children at each machine, for a total of about 6 hours..

Activities
Across all 56 children the activities were very varied, since they were all present for differing lengths of time. However, the children whose data is presented here were all given relatively structured activities to perform, with the focus being on the writing and comprehension of rules.

Thus, for example, they were asked to write a rule to move an agent to the right (or left), and a rule to enable the agent to climb over an obstacle, etc. As well as being given specific rules to write, the children also were given opportunities to create their own rules for their own agents.

Besides the on-line KidSim™ activity, the children were also given a pencil-and-paper test of their understanding of rule construction in KidSim™. This contained monochrome images of some simple graphical rewrite rules, which the children had to write a one sentence description of. These rules were in fact much the same as the ones we asked them to write, but they included some important characteristics (for example, two versions of the same movement, but one with ground beneath the agent and one without).

Results
Our studies did not aim to produce clean readily analysable data - partly due to the speed and unreliability of the KidSim™ system anyway. However, from the videos it is possible to extract substantial quantities of information concerning their comprehension and their enjoyment.

Ease of Use
A striking feature of the children's activities is their overwhelming enjoyment of using the system. This was more true of the 11/12 year olds than the 13/14 year olds, but there was a strong sense of disappointment at the end of each session and an eagerness to return.

In the time when the children were able to create their own worlds and agents it was clear that KidSim™ was a spur to their creative imaginations. An enormous variety of different worlds were created, ranging from wars and battles, to aquariums or soccer pitches! However, there were few differences in the rules written for these different worlds – although the agents concerned were very different, their programmed actions were surprisingly similar.

It appears, therefore, that KidSim™ does provide an environment where children have the opportunity to learn about programming at the same time as solving their own problems, rather than teacher-defined ones.

Hardly any of the children had any major difficulties in using the interface to construct agents and rules. Across all their sessions (40–80 minutes) the children constructed an average of 8 rules per session (3–15). Some of these were rules suggested by the us, whilst others were of their own invention.

The most common problem was over-eager mouse-clicking due to the slow responses of the system. This, coupled with a problem in the rule construction process (due to our use of 16" monitors), led to a number of "dud" rules that did absolutely nothing.

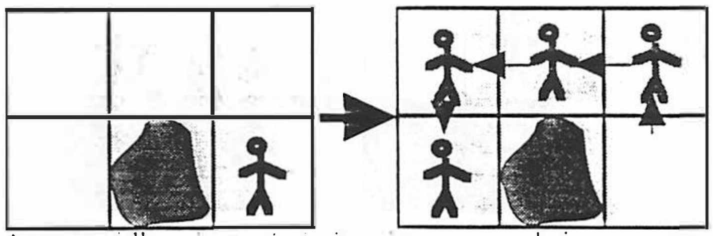
Figure 1: A successfully written rule for jumping over a rock, incorporating 4 actions.

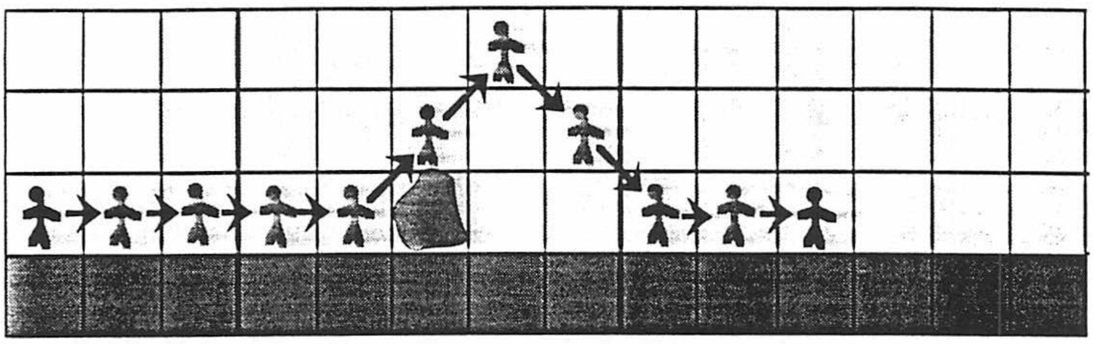
Figure 2: A similar rule using an unnecessarily large spotlight (e.g. the 12 squares at the right end) and 10 separate actions. Such a rule could be labelled an animation.

Rule-Writing
Ignoring some minor interface difficulties to rule-writing, it is the choice of rule and its implementation which gives us much information about the children's understanding of programming.

A rule in KidSim™ is defined by a spotlight around an agent which indicates the scope of matching required before the rule can fire.

A key question, therefore, is what size of spotlights did the children prefer to use? A small spotlight indicates a comprehension of the matching process and the general model of repetitive rule application. Likewise one can ask about how many actions an agent makes within a single rule, since a more useful, generic rule (containing a single small action) would seem to reflect a greater degree of understanding than large rules containing many actions.

For example, Figure 1 illustrates a rule successfully written to make an agent jump over a rock., whereas Figure 2 shows a different pair's attempt at the same rule. This second rule shows how some of the children, at least, had a model of the system which did not distinguish between single and repeated rule firing. Their rules resembled animations rather than programs for action.

Figure 2 also illustrates how some of the children included squares in the spotlight which were not used by the animation. These redundant squares limit the applicability of the rule and suggest that the children do not have a good model of the matching process. In fact, their discussions with each other suggest that they have a model in which empty squares are irrelevant, whereas in fact, KidSim™ will only match an empty square with an empty square.

Summarising over the children reported here, we found differences in a variety of measures between when the children were doing our exercises and when they wrote their own rules.

The average spotlight size when writing rules to meet our specifications is 5.5 squares, but when writing their own rules, this increases to an average of 36 squares (ranging from 6 to 80). Likewise the number of redundant squares in the spotlight averages 2.5 on exercises and 14 on their own rules. It is interesting to note, however, that this is approximately the same proportion of redundancy in each case (approximately 40%). Finally the number of steps in each rule averages 1.1 for the exercise rules, but 3.1 for their own rules.

Rule Comprehension
The average score on the test of rule comprehension was 7/10, with the majority of marks being lost on the three questions which involved spotlights which included pieces of ground. Figure 3 shows an example of one of these questions, which is almost

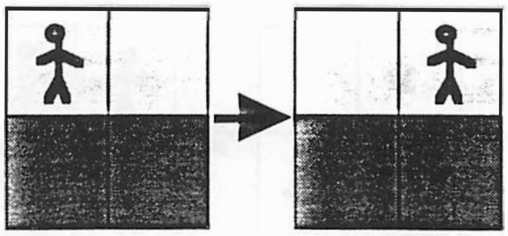

Figure 3: Question 2 from the rule comprehension test, for which only 22% of children mentioned the importance of the ground.

identical to the test item preceding it, except for the presence of the ground squares.

A further means of studying rule comprehension is to examine the set of rules written for its coherence. What we observed here is that the children seem to spend very little time reading back over the rules they have already written for an agent. Indeed, one pair of children, who initially appeared very productive (15 rules written in just over one hour), turned out to have been producing multiple copies of very similar rules. Of the 15 rules written, only 6 can really be regarded as distinct rules (a rule to move right four spaces was written 4 times).

Similarly, on occasions where a rule did not work as expected, the children showed no awareness of the concept of debugging. In most cases, the reaction to a rule which did not work as expected was an attempt to rewrite the rule from scratch.

Confirmation of the animation model of KidSim™ also comes from some the dialogue which occurred when things did not go as expected. A quite common utterance was of the form "But we didn't write that rule for there.", indicating that they expected a rule to apply in the context where it was written and not simply anywhere that matched.

Qualitatively

As already mentioned the children enjoyed their KidSim™ activities and, despite some of the lack of comprehension, they produced some surprisingly interesting worlds. These worlds were of great interest to the children, not just to us.

For example, there were a number of occasions where the children demonstrated their worlds to other children, along with discussions about how various effects had been achieved.

One of the pairs actually took control of the video camera and constructed their own short demonstration of KidSim™ and how to use it. In this video they actually make jokes about the speed of the program, but still manage to offer a sales-style presentation.

Another indicator of the children's enthusiasm, and one which surprised us, was their willingness to reconstruct worlds which were lost when the system unexpectedly crashed. Some of the children had 2 or 3 attempts at constructing the same large world (up to 30 minutes each time) which was lost due to unreliability. Whilst it would not be true to say that they did not complain about having to rebuild the world, the agents and the rule-sets, they set about the reconstruction with remarkable patience.

Summary

The children understood how to use KidSim™ and in many cases were not aware of their lack of comprehension. They became actively and enthusiastically involved in constructing their own worlds, agents and rule-sets and it is clear that KidSim™ was a great spur to their imaginations.

However, there is plenty of evidence to suggest that they really didn't have any depth of understanding about the programming concepts involved. Many children expressed puzzlement over the use of the word 'rule'; many of them wrote rules which appeared more like animations (cartoons) than rules; few of them showed any clear understanding of the graphical matching process.

DISCUSSION

The implications of these results are mixed, since it would seem that KidSim™ has many features which the earlier LOGO environments lacked and which may be conducive to learning about programming. But, at the same time, we haven't obtained any good evidence that the programming concepts have been learnt.

This gives rise to two possibilities:-

a Children would acquire more programming knowledge if KidSim™ were redesigned.

b. Children / end-users cannot acquire these programming abstractions.

Possibility (a) is our currently preferred conclusion, since the system used in the evaluation had so many weaknesses. For example, the system was slow and crashed regularly, there was no documentation and the main research staff did not have much knowledge of the detailed workings of the system.

However, it is not our belief that the redesign should make the interface any easier, since children seem to be successful with it already. The problem seems to be more one of the interface being too easy and, thereby failing to encourage reflection and learning (see Gilmore, 1994).

Possibility (b) may be true for situations where there is no scaffolding for the acquisition of these concepts, or where the interface / system reliability obscures them. The design challenge is to make the system support the learning of the programming abstractions, whilst maintaining the acceptability of the user interface, where the former will almost inevitably add more complications to the interface and the user's model of the system.

Analysing our data for the problems which arose, and which may have prevented programming concept acquisition, we generated a list of over 20 possible features to redesign. However, almost all of these changes have side-effects on the others.

The goal was a system in which interface and functionality encourage children towards 'small spotlight, small action rules', not 'large spotlight, multiple action animations', in the belief that without the former the abstractions will not be learnt.

In the end three changes were selected which were felt to be achievable within the available time-frame and were also thought to have the desired effects without unwanted side-effects:-

1. Speed improvements. These were inevitable anyway, but one reason for the children's preference for animations may have been the speed at which the rule editor appeared (once open, they felt obliged to use it for as much as possible);

2. Rules apply to all agents of the same type. Currently a rule belongs just to the agent it is created for and, therefore, it is not surprising if children do not appreciate the generality of the rule. If rules apply to all similar agents in the world, then there is a big cue to the importance of generality.

3. Individual agents can be saved and imported into new worlds. Along similar lines it can be argued that enabling agents to be moved between worlds means that it is important to consider how one's rules will work in an, as yet unknown, world.

The advantage of these changes is that they should maintain, or even increase the acceptability of the system (since rule writing is faster, applies to more agents and may not always be necessary) to children, whilst at the same time providing clear scaffolding for the development of programming concepts.

SUMMARY

It is possible to offer a graphical programming environment to young children which enables them to address their own problems and interests rather than those of a teacher. Whether they acquire the generalised programming constructs and thinking skills from such a system is as yet untested, but the enthusiasm with which the KidSim™ system was received suggest that this will provide a very good test bed for addressing this issue.

On the basis of our results, KidSim™ can be labelled an end-user programming environment. The long-term importance of this is that if it does indeed prove possible to scaffold the acquisition of programming concepts in 11 year old children, then the prospects for genuinely powerful end-user programming languages are extremely promising

ACKNOWLEDGEMENTS

This collaborative project between Nottingham and Apple Computer Inc. was supported by a NATO Collaborative Research Grant (no. 920469).

REFERENCES

Gray, W. D., Spohrer, J. C., & Green, T. R. G. (1993). End-user programming languages: The workshop report. SIGCHI Bulletin, 25(2), 46-50.

Kurland, M., Pea, R. Clement, C. & Mawby, R. (1986). A study of the development of programming ability and thinking skills in high school students. *Journal of Educational Computing Research, 2(4)*, 429-458.

Linn, M. & Dalbey, J. (1985). Cognitive consequences of programming instruction. *Educational Psychologist, 20(4)*, 191-206.

Mayer, R, Dyck, J. & Vilberg, W. (1986). Learning to program and learning to think: What's the connection? *Communications of the ACM, 29(7)*, 605-610.

Papert, S. (1970). Teaching Children Thinking. *Mathematics Teaching, 1970.*

Smith, D.C. Cypher, A & Spohrer J. (1994). KidSim: Programming agents without a programming language. *Communications of the ACM, 1994 (July)*, 54-67.