

# Degrees of Comprehension: Children's Understanding of a Visual Programming Environment

Cyndi Rader, Cathy Brand and Clayton Lewis

Department of Computer Science

University of Colorado

Boulder, CO 80309

(303) 492-4932

crader@cs.colorado.edu, brand@cs.colorado.edu, clayton@cs.colorado.edu

## ABSTRACT

A new generation of innovative, highly visual children's programming environments is under development. In this paper, we consider the instructional requirements for children learning to program in a visual environment. Based on our year-long experience using Apple Computer's KidSim/Cocoa prototype [2] and the results of a year-end assessment, we conclude that the children failed to grasp many aspects of the program operation. The children readily mastered drawing and animating characters in imaginary worlds, but struggled to achieve more complex behaviors. Lack of explicit instruction on program functionality hindered these children in their attempts to create more sophisticated science programs. We explore the prospects for more effective instruction and suggest some guidelines for designing visual programming environments.

## Keywords

Kids Software, educational application, end-user programming, simulations, programming by demonstration, graphical rewrite rules

## INTRODUCTION

Teachers and parents often value the time that elementary school children spend playing math drill games and using word processors because they believe that children should learn to use computers. Can we make the more creative aspects of computer usage available to young children as well? A new generation of innovative children's programming environments is emerging, including KidSim/Cocoa [2] and Toon Talk [6,7]. Will these new, highly visual environments succeed in involving children in programming?

KidSim/Cocoa differs from preceding children's programming languages, such as Logo and BASIC, in that children "program" in this environment through direct manipulation of pictorial objects. The goal is to provide a very expres-

sive medium for children to engage in creating their own simulations[2]. The designers believe they have eliminated many of the syntactic and semantic issues that make conventional languages difficult to learn and use. Abstraction is easier to comprehend because children can create abstract rules by concretely demonstrating what objects should do.

If children are to learn to program in KidSim, what kind of instructional environment is needed? Will they thrive on relatively unstructured individually-paced experiences, as with LEGO™ construction toys, or will they need highly structured experience of the kind a school setting might provide? Previous researchers have demonstrated that students need well-designed instruction over a long period of time, as well as certain cognitive skills more commonly associated with the middle school years, to progress beyond rudimentary programming in a traditional programming language[10]. Have the new children's languages broken through these barriers to usability?

This paper reports data and experience from the Science Theater/Teatro de Ciencias (sTc) project in which a 2nd/3rd grade bilingual class and a 4th/5th grade class worked with Apple's KidSim (now called Cocoa) prototype as part of their science curriculum. We focus on an assessment in which we asked the children questions about how KidSim programs work. The results suggest clear limitations on what the children learned about the software and some reasons for these limitations. We relate these results to evidence from programs the children created to form a more comprehensive picture of what was simple and difficult for the students to grasp about the KidSim programming environment.

We conclude that younger elementary school children face significant challenges in mastering the KidSim environment well enough to create programs useful in their science study. Older children fare better, but also often fail to grasp important aspects of KidSim. In our discussion we consider how these challenges influence what educational value programming can deliver for kids. Although others have argued the benefits of learning programming for its own sake

Permission to make digital/hard copies of all or part of this material for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copyright is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires specific permission and/or fee

CHI 97, Atlanta GA USA

Copyright 1997 ACM 0-89791-802-9/97/03 ..\$3.50

[6,9], our primary interest is in programming as a vehicle to support conceptual development in other areas. With this end in view, we relate our results to three possible scenarios for children using KidSim to explore topics in science. We conclude with implications for both the user interface and the type of instruction.

### BACKGROUND

KidSim is a grid-based, graphical rewrite system that allows children to create simulations using direct manipulation and programming by demonstration [2]. Figure 1 shows two rules: the top one makes the raindrop move down (a move-down) and the bottom is a move-right. These rules are created by "showing" the object what to do (e.g., the top rule would be created by dragging the raindrop to the grid square directly beneath it). The behavior is recorded as "before" and "after" pictures.

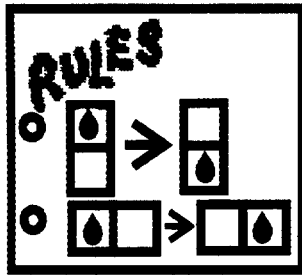


Figure 1: KidSim Rules

Because the programming environment seemed quite transparent and our goal was for students to explore science content, not to learn to program, we decided not to provide explicit instruction on the mechanisms underlying KidSim. We provided a very minimal demonstration of the system, showing them how to draw a new object and how to create a rule to make the object move to the right. The students then worked alone or in pairs to create their own "exploratory" worlds. They spent about 45 minutes a week for 6 to 8 weeks creating characters and making them move around in an imaginary world.

To expose students to additional features of the software, researchers suggested enhancements to the worlds and guided the students through using the required features. For example, one boy created a world with a truck traveling across the screen. He then wanted the truck to "roll" over a rock, which required learning to include context (the rock) in rules. He had the truck stop in front of a store (more context), a man get out of the truck (create rule), walk to the store (move rule) and go into the store (vacuum rule). He kept a counter of the number of people in the store (property rule), although he never did anything with that property. One girl did use a property to control the number of times her dancer would jump on a trampoline before starting to do cartwheels. The effect of cartwheels was created by changing the appearance of the dancer as she moved across the screen. A few students used subroutines to make their objects (e.g., eels) move randomly around the screen. Because we put no specific requirements on these exploratory worlds, there was wide variability in what fea-

tures individual students used. Some students, especially in the younger class, spent most of their time drawing.

In the second phase of our year-long curriculum, students used KidSim to build graphical models that explained various scientific phenomena. They chose topics from the science units covered by the classroom teacher. The 2nd and 3rd graders explored questions such as "Would flowers grow in space?" and "How were Egyptian mummies created?". Topics investigated by the 4th and 5th graders included the spread of skin cancer and the formation of blood clots. The students usually worked in pairs to create their science models with two researchers facilitating groups of 6 or 7 children. This phase lasted about 8 weeks, with students again working 45 minutes per week.

### UNDERSTANDING PROGRAM OPERATION

One benefit of the KidSim environment is the ease with which simple worlds can be created. Within minutes, students can learn to create rules that make objects move about the screen. However, since our primary goal is for students to use KidSim to explore science topics, they will need to do more than assign arbitrary behaviors to pieces they create. Building models requires students to decompose a problem into a set of objects with particular behaviors. For a model to "show" a phenomenon, the students must figure out how to map the desired behaviors onto the KidSim language capabilities. They must also consider how to distribute behaviors among the various pieces to achieve a global program operation that matches the science concept or phenomenon of interest. In this section we describe some aspects of the KidSim program that students will encounter as they build fairly sophisticated worlds. In a later section we will describe how the students were able to use the KidSim features in their science models.

Designing, implementing and debugging complex worlds requires some knowledge of the underlying operation of KidSim. Students will need to understand the following:

1. Individual actions. Understanding the action of a rule requires comparing the "before" and "after" pictures to determine the action. For example, in the top rule in figure 1, the drop is in the top square before, the bottom square after, which translates to a "move-down" rule. Sometimes the script created during the programming by demonstration session that created the rule must also be examined.
2. Rule ordering. The rules for each object are evaluated in order from the top of the list during each time cycle. The first rule that can fire (based on "before" picture and preconditions such as properties) will fire.
3. Picture matching. For a rule to fire, the conditions in the "before" picture must be matched exactly. The matching routine considers the object's current appearance as well as the existence and appearances of any

other objects included in the rule. Empty spaces are also significant.

4. Object Interaction. Objects interact as they move around the grid either directly, by including multiple objects in a rule, or indirectly, which occurs when one object "blocks" another's movement.

KidSim also includes some features which are useful for programming more complex behaviors:

- Subroutines. Rules may be grouped and performed either sequentially or randomly.
- Properties. Objects can have properties (variables) that can be updated and checked in rules.

## METHOD

To assess how well the students understood program operation, we developed a series of 10 tasks that used the basic features of the KidSim programming environment. The assessment included three different types of questions: predictions (e.g., what will happen when we run the program), explanations (e.g., why are the objects not moving) and corrections (e.g., how can we get this object to move). The students were told at the beginning of the interview that they should pay attention to the rules. The general format of each task was to display the rules and ask the student to predict what would happen. The interviewer would then "run" the world to check the prediction, then ask some number of explanation and/or correction questions.

In summarizing the data we relate each task to the aspects of KidSim listed above. Table 1 shows the purpose of the task and gives the percentage correct for the prediction, explanation and correction. Not every task includes all three types of questions. In interpreting the results, note that confidence intervals for proportions depend upon the sample value as well as the sample size. For both groups the sample size was 20. A 95% confidence interval for a sample percentage of 50% extends from 27% to 73%. For a sample percentage of 5% the interval extends from near 0 to about 25%. Confidence intervals for differences of these percentages also depend upon the sample values. The half

width for a comparison of 50% vs. 90% (or 50% vs. 5%) is about 25%, so such a difference is significant at the 95% level while a difference of 50% vs. 75% is not. Significant differences between the grades will be indicated by an asterisk (\*) in the 4th/5th grade column.

## INTERPRETATION OF 10-TASK ASSESSMENT

In this section, we explain the tasks in more detail and interpret the results of Table 1 with respect to the knowledge required to answer each question correctly.

### Individual Actions

The first task included a single raindrop with one rule, a move-right. 20% of the 2/3 and 60% of 4/5 students correctly predicted the raindrop would move right. When students were specifically directed to look at one move-right rule and state what that rule would do, 56% of 2/3 and 90% of 4/5 students correctly identified it. When directed to look at a move-down rule (task 2), 53% of 2/3 and 60% of 4/5 correctly identified it. A common mistake for the second task was to interpret the action as a move-diagonally (right and down), presumably because they looked at the pattern presented by the entire rule, rather than analyzing the difference between the before and after pictures. Several students answered "up and down," which might indicate recognition of the fact that objects wrap around the screen. This answer might also be attributed to the imprecise use of language common to students of this age.

### Rule Order Comprehension

Task 3 consisted of a move-down rule above a move-right (as in Figure 1). Although the majority of the 4/5 students understood the individual rules, only 5% could correctly predict that the drop would move down. To answer correctly, students would need to understand that the program always starts testing the rules from the top during each evaluation cycle. The most common answers (74%) seemed to indicate that the program would use some combination of the rules, such as alternating between them. The younger group was less able to interpret the individual rules

Task	2nd/3rd			4th/5th		
	Predict	Explain	Correct	Predict	Explain	Correct
1. Individual Action - Right	20	56	-	60*	90*	-
2. Individual Action - Down	-	53	-	-	60	-
3. Rule Order	30	-	5	5	-	15
4. Rule Order/Action	10	29	-	15	70*	-
5. Picture Match - Appearance	11	15	35	15	68*	80
6 Object Interaction	5	20	-	5	45	-
7. Object Interact	63	10	-	88	50*	-
8. Sequence Subroutine	15	-	-	18	-	-
9. Random Subroutine	17	0	-	50*	75*	-
10. Property	0	41	39	0	47	55

Table 1: 10-Task Assessment Results. \* indicates significance.

(56% and 53%), but did slightly better predicting the overall behavior (30%). They may have based their predictions on the behavior expected of a real raindrop which, in this case, was duplicated by the programmed raindrop.

After seeing that the raindrop moved down, students were asked how to get the bottom rule to run. Performance by 2nd/3rd graders fell to 5% on this task, indicating either that they did not really understand rule ordering or possibly that they understood it but did not know how to change the order of the rules (a drag-and-drop operation). For the older group, the percentage was 15%, a discouraging result given that understanding rule order is crucial for achieving correct program behavior in many instances.

Task 4 tested both rule ordering and individual action. The top rule was a do-nothing rule (matching before and after pictures). These rules are created whenever students start to make a rule and then press *Done* before doing any actions. Recognizing these rules is therefore important for debugging. In task 4, very few students correctly predicted that nothing would happen, although 70% of the 4/5 students were able to explain why afterwards. Only 28% of 2/3 students noticed that the top rule had no action.

#### Picture Matching

Task 5 tested students' understanding of the role of appearance in pattern matching. The world included a "smiley" face, a "sleepy" face, and a rule for the smiley face to move right. Very few students in either class realized that only the smiley face would move. After seeing the result, however, 68% of the 4/5 were able to explain it and 80% were able to get the sleepy face to move by changing its appearance. In the younger class, 35% were able to make the face move. We noticed that some of the younger students knew how to do an operation within the program, but didn't have complete knowledge of when or why to do it. Changing appearances was a common action for these students, so it is possible that observing the two appearances triggered that knowledge, even though they didn't understand the connection between the appearance and the rule not firing.

#### Object Interaction Task

Two tasks checked students' understanding of how objects interact. In task 6, two large smiley faces were placed close together on the grid. Because the move rule for the smiley face required empty space around it, neither face could move. Only 1 student in each class made the correct prediction, and most could not give explanations.

In task 7, a raindrop moved down until it was in the grid square above a rock. Most students were able to predict that the raindrop could not keep going in the same direction (it's not clear that they were using knowledge of the program to make that prediction). However, only 10% of 2/3 and 50% of 4/5 knew what the raindrop would do. Since

the raindrop could not fire the first rule but could fire the second, it would execute the second rule (a move-right). This task required fairly complex reasoning. However, if students create simulations of interacting objects, they will sometimes encounter these types of scenarios.

#### Subroutines

Task 8 included four rules in a sequence subroutine which caused a fish to move in a circle. Task 9 used the same rules in a random subroutine. About 15% of 2/3 and 48% of 4/5 showed some understanding of sequence routines. 17% of 2/3 and 50% of 4/5 understood random routines.

#### Properties

Task 10 required that students understand how properties affect rule firing. None of the students initially noticed the property, which is not surprising since there are no visual cues about the property in the rule pictures. After prompting, 39% of 2/3 and 55% of 4/5 were able to manipulate the property in order to change the behavior of the world.

#### SUMMARY OF 10-TASK ASSESSMENT RESULTS

One aspect of KidSim that underlies every task is the fact that program behavior is controlled by the content of the rules. In spite of explicit instructions to look at the rules when answering questions, many students showed a tendency to guess, either relying on 'real-world' reasoning or past experience with the program. For example, students predicted that a raindrop would fall or a sleepy face would wake up although the rules did not in fact show these behaviors. Students in both age groups gave such answers, but this behavior was more common in the younger group. The younger group also appeared to base predictions more on past behavior of the program. For example, if a raindrop in one world moved down, they had a tendency to predict that it would fall again in the next world.

To summarize the results, it appears that the majority of the students in the 4th/5th grade class understood:

- that rules control behavior,
- the meaning of simple (move) rules,
- the role of appearances in controlling behavior,
- the concept of randomness, and
- how to manipulate properties.

Less than half of them understood:

- how rule ordering works,
- exactly how objects on the grid interact (although they knew that they do interact), and
- sequence subroutines.

From the analysis of models they created, we can also conclude that very few had any idea how to use properties.

We conclude that the 2nd/3rd grade students understood:

- that rules control behavior (but they are not always clear exactly how), and

- the meaning of simple move rules.

Less than half of the students seemed to understand the other aspects listed above.

Based on these results, it is clear that the students could not acquire a complete understanding of the underlying operation of KidSim in the instructional environment we created for them last year. Some possible explanations for their misconceptions include:

- Experience-based reasoning. As mentioned above, students tended to believe that objects would behave as they do in the real world.
- Anthropomorphism. Children seem to expect the computer to match pictures the same way they do. For instance, they would observe that one object is in front of another, but not notice specifically how many grid spaces separate the two objects or if one object is also slightly higher or lower than the other one.
- Holistic perspective. Children of this age are not used to decomposing a scenario into actors with behaviors. For example, one student initially created a single piece that included the sun, the ocean and a boat.
- Narrative perspective. Children also tend to visualize a process as a sequence of pictures. Although they can create engaging animations in this way, they miss some of the benefits of doing simulations.
- Lack of precision in descriptions. Younger children in particular will say they want an object to move around rather than designate a direction. They also struggle to specify exact sequences of events. For instance, the younger children could not always identify repeating patterns such as a fish moving in a circle.

### Evaluation of Student Worlds

The 10-task assessment measures comprehension of KidSim rules but not production of KidSim worlds. To find out how the students actually used the program, we counted the number of features of each type in their worlds. These data provide some indication of which features were most useful and how well students could map desired actions in their models to KidSim capabilities (e.g., using create rules for having a baby, properties to count sun rays, appearance changes to show flowers growing, etc.).

Recall that students had two experiences creating worlds: an open-ended exploratory world and a model of a scientific phenomenon. Table 2 lists each feature and provides a comparison of how often that feature was used in the exploratory versus scientific worlds for each grade. The table includes both the number of student groups that used the feature and the total number of times the feature was used. Note that the students were assisted by researchers in creating their worlds, so these data cannot be taken as a direct proof that the students themselves understood every programming construct they used. The table includes data

from 23 2nd/3rd exploratory worlds, 22 4th/5th exploratory worlds, and 17 science models from each grade.

In both exploratory and science model worlds, the 2nd/3rd graders used mostly move rules and a few appearance changes. The easiest action for these students was simple forward motion. A few students, with researcher assistance, managed to get their characters to move randomly. The biggest difference between the science model and exploratory worlds is that the students created more rules during the exploratory phase. In fact, 6 of the 17 science models contained no rules at all. Although the themes for the exploratory worlds were simpler, they were often worked out more completely than the science models, possibly because the students were not constrained by specific content requirements during the open exploration. The process of building science models required students to come up with a question that could be modeled, research the answer to the question, then plan and implement a model. We had hoped that they would learn more KidSim concepts through the process of building models, but dealing with all of these challenges prevented students from further exploring the language.

In contrast to the younger class, the 4th/5th graders successfully constructed both exploratory and science worlds. Their exploratory worlds consisted primarily of moves and appearance changes. The science models relied more on create and vacuum rules which allowed more elaborate scenarios in which objects appeared and disappeared. A few groups who had not used subroutines earlier were able to incorporate them into their science models. Only one student used a property correctly in his science model and he required help from a researcher. The 4th and 5th graders were able to use more of the features as of KidSim as they needed them. However, most of them could not use subroutines and properties without additional instruction.

Another difference between the two age groups is the total number of rules used. All of the science models created by the older students had rules, compared to only 11 of 17 in the younger class. Even the 2/3 models which did have rules generally had less rules than the older group. The total number of features used by the 4/5 students was 278 with a mean of 16, compared to 73 features with a mean of 4 for the younger students. In the exploratory worlds, the total number of features for 4/5 was 298 with a mean of 13.5, compared to 128 total features with a mean of 5.6 for 2/3 students.

### DISCUSSION

The results of the 10-task assessment illustrate students' understanding within the context of answering questions or making predictions about worlds presented to them. Here we are interested in the relevance of our results to 3 different scenarios: students building their own worlds, students

Feature	2nd/3rd Grade				4th/5th Grade			
	Exploratory		Science		Exploratory		Science	
	# Grps	Total	# Grps	Total	# Grps	Total	# Grps	Total
Move	22	95	8	59	21	186	17	90
Appearance	7	22	2	3	11	72	16	83
Create	0	0	2	5	2	2	13	44
Vacuum	4	5	3	3	7	18	12	41
Property	1	1	0	0	6	6	1	3
Sequence	2	2	1	2	6	8	4	13
Random	3	3	1	1	5	6	2	4
<b>Totals</b>	<b>39</b>	<b>128</b>	<b>17</b>	<b>73</b>	<b>58</b>	<b>298</b>	<b>65</b>	<b>278</b>

**Table 2: Usage of Features in Student Worlds**

revising their own worlds, and students revising worlds that other people have created.

#### **Building new worlds and models.**

An obvious goal of children's programming environments is to enable them to create their own worlds. All of the children enjoyed drawing colorful characters and making them move around an imaginary world. They proudly displayed their creations to their classes at the end of the exploratory period. During the second phase, many of the younger students asked repeatedly to return to free creation.

It is possible to create a world without much knowledge of the underlying program mechanisms. Although such worlds might be simple from a programming perspective, they can be quite appealing visually. For instance, during the exploratory phase, several students made a world which showed a skateboarder doing flips. The combination of realistic movements and simple but attractive drawing made the world entertaining to watch. Programming by demonstration enables children to "program" a simple cartoon-like world but have no awareness that they are programming. Because rules are created in the order that actions are performed, the desired behavior is often achieved with no explicit consideration of rule ordering. Children can readily imagine a story and create it, one scene at a time. Furthermore, we observed that when the students are creating an imaginary world, if they can't program some desired behavior, they simply do something else.

Ultimately we question how much students would learn if they never progress beyond this stage. Once students have learned a few program mechanics, they could create numerous worlds of the type just described. As others have noted, children easily learn the primitives of a visual environment and then struggle to program with them in interesting and complex ways [7].

A primary focus of the sTc project is to encourage students to develop deeper explanations of scientific phenomena by creating explanatory models. Such explanations often require representation of causal mechanisms and their influence on or interaction with other objects. For example, in a model of sunburn/tanning, ultraviolet rays are a causal mechanism. An explanation of tanning might show the stimulation of melanocytes by the UV rays. Developing such a model will normally require a greater degree of programming sophistication, because the objects will have their own behaviors which may vary based on their interaction with other objects. As objects interact, both rule order and pattern matching become more important. More complicated behaviors, such as random motion or modeling the effects of time, require use of subroutines and properties.

#### **Revising their own worlds and models**

It might be argued that performance on the 10-task assessment is not an accurate indication of students' abilities to revise their own worlds, since it required students to suggest changes to worlds that they did not create. However,

in a survey asking students what was most difficult for them, ordering rules to get the models to work correctly was the most frequent response. We also observed that students tended to resist suggestions to revise or extend their science models. Many students felt that it was just too difficult to make changes.

For example, a 5th grade student created an explanatory animation of how life began. He had a mental image of clouds of chemicals blasting out of a volcano, mixing together and being zapped by lightening to create the chemical compounds which function as the building blocks of living cells. Then these proteins would join together and form single cells. After a long passage of time, the single cells would join together into cellular blobs, which would gradually take on different functions and develop into simple animals. This student explained the process in detail to his class, emphasizing the amount of time that passed before simple animals appeared. Yet his computer model skipped rapidly from clouds of chemicals to slugs. Once the student had a working model, even though it left out many interesting and important details, he was reluctant to make any changes to it. Since he treated all the atoms of a chemical as a single piece (i.e., a cloud of oxygen atoms, etc.), it was difficult to combine the chemicals in various patterns during the lightning strikes. He struggled both to create pieces of the right grain size and to come up with specific steps that could map to rules.

An important component of understanding science models is realizing that they can and often should be revised. Students creating models will need to revise them when they discover new information or when they want to extend their explanation. If students are using their models to speculate about causes, they may need to revise their models as they discover flaws in their initial conceptions. In order for the students to focus on the science content, revisions must easily be accommodated by the programming tool.

#### **Revising other peoples' worlds and models**

Within our environment, the emphasis is generally not on revising worlds created by other people. We can envision, however, that sharing worlds may be an enjoyable extension to working with the program. The 10-task assessment bears most directly on the students' abilities to modify worlds which are provided to them.

Modifying code written by another person can often be challenging in ways that have little connection to the ability to modify one's own code. In the 10-task assessment, however, the worlds were extremely simple, with most having only one or two rules. Because of the simplicity of the rules and lack of science content, we feel that many of the obstacles to modifying another person's code (i.e., the need to understand how variables are used or to follow the control structure of the program) were reduced or eliminated. Nev-

ertheless, performance was poor. Without a better grasp of the program operation, we believe these students would encounter serious difficulties in modifying worlds that they create, as well as those that are provided to them.

#### **Suggestions for a More Effective Visual Environment**

Our results have implications for the design of visual program environments. When learning to program is not the ultimate goal, it is particularly important for the interface to support users in performing typical tasks. If an understanding of the underlying program mechanism is not necessary for performing common tasks, then the user should be insulated from those details. On the other hand, if having such an understanding is required, then the environment should provide as many cues as possible toward that goal. As Gilmore et al. note [4], if the interface makes rule creation too easy, children focus on making lots of rules without pausing to reflect on how the whole system functions. We believe that students creating realistic science models (or any worlds with prespecified and fairly precise behaviors) need to have a reasonably complete understanding of how the program operates in order to plan and generate the desired results. Based on this assessment, it seems clear that students had many misconceptions about the program. Some issues to be considered include:

1. How to translate a programming-by-demonstration action into a visual representation that is easily recognized and understood. It can be quite difficult to look at two pictures and quickly determine the difference between them. We have noticed informally that it is harder to recognize a move-left rule than a move-right. We speculate that this is because of our cultural bias to read from left to right and possibly because there is a right-pointing arrow that separates the before and after pictures. A problem with using two pictures as the sole representation is that there is no way to show intermediate steps. It is possible for the before and after pictures to be identical, but for the rule to have some action that ultimately returns the object to its starting state. A "play this rule" button would be helpful to see what the rule actually does.
2. How to quickly determine whether all the conditions exist for a rule to fire. It is quite frustrating when a rule with seemingly true conditions will not fire. In Visual AgenTalk, a rule can be tested by dragging it on top of an object [11]. This action provides a useful indication of whether the rule could fire, but does not give any clue as to why a rule can't fire. Highlighting the condition that is not met (e.g., an incorrect property value, or overlapping items that should not be in the same space) would be extremely helpful.
3. How to make the rule selection process explicit. Most students have no concept of what happens during a single time step. Adding visual cues to the rule window (e.g., a column called "Check this rule..." with "1st" etc. listed beside the rules) might help. Allowing

the users to experience the action, such as with a button that causes the current object to simulate the rule selection process, might be even more effective.

4. How to make the object list explicit. When two interacting objects have rules, the ultimate behavior often depends on which object checks its rules first. There are no visual cues that provide the ordering of objects.

#### Plans for More Effective Instruction

Regardless of how well the interface is designed, we expect that users will need additional support to learn to perform complex tasks within this environment. Without an understanding of the underlying mechanisms, it would be difficult to do more than guess at program behavior and randomly use program features when trying to solve problems. The 10-task assessment provides clear indication of that behavior by some of the students. Other studies have indicated that when time is limited, a more structured approach is required to teach programming ([5] cited in [8]). For this year, we have developed a structured set of "challenges" to introduce the students to the main software features and to provide some of the guiding knowledge that we have distilled about the system[1]. Since mastery of the software is not enough by itself, we will also be using a variety of activities, including work with precreated models and classroom discussions, to help students learn the other aspects of model creation. Preliminary results with both the challenges and extra activities have been encouraging.

When students struggle to master the syntax of a language, they often ignore the semantics ([3] cited in [8]). We would like the model-building process to encourage students to explore their ideas in more depth. What we saw more often last year was that students had sufficient difficulty in representing their current ideas that they were not encouraged to pursue any deeper understanding. Given the lack of explicit teaching about both the underlying program operation and the "pragmatics" of building models, we were actually pleased that students made as much progress as they did. We are optimistic that, with sufficient explanation and exposure, students will be able to master the most important aspects of the program and build more complex and scientifically interesting models.

#### ACKNOWLEDGMENTS

The authors would like to thank the entire sTc team: Mike Eisenberg, Nancy Songer, Teresa Garcia, Linda Hagen, Heidi Carlone, Carlos Garcia, Page Pulver and Cecil Robinson. KidSim© was designed and developed by David Smith and Allen Cypher of Apple Computer. Our project is funded by the National Science Foundation's Applications of Advanced Technology program.

#### REFERENCES

1. Bell, B., Citrin, W., Lewis, C., Rieman, J., Weaver, R., Wilde, N., and Zorn, B. (1994) Using the Programming Walkthrough to Aid in Programming Language Design, *Software-Practice and Experience*, 24(1): 1-25.
2. Cypher, A. and Smith, D.C. (1995) KIDSIM: End User Programming of Simulations, in *Proceedings of CHI '95* (Denver CO, May 1995), ACM Press, 27-34.
3. Fay, A.L. and Mayer, R.E. (1988). Learning LOGO: A Cognitive Analysis, *Teaching and Learning Computer Programming: Multiple Research Perspectives*, R. E. Mayer (Ed.), Hillsdale, NJ, Lawrence Erlbaum Associates: 55-74.
4. Gilmore, D., Pheasey, K., Underwood, J. and Underwood, G. Learning graphical programming: An evaluation of KidSim, In *HCI: Interact '95 Proceedings of Fifth IFIP Conference on Human Computer Interaction*, Lillehammer, Norway, June 1995.
5. Heller, R.S. (1986). Different Logo Teaching Styles: Do They Really Matter, *Empirical Studies of Programmers*, E. Soloway and S. Iyengar (Eds.). Ablex Publishing Corporation, Washington, DC, 117-127.
6. Kahn, K. (1996) Toon Talk™ -- An Animated Programming Environment for Children, *The Journal of Visual Languages and Computing*, June 1996.
7. Kahn, K. (1996) Seeing Systolic Computations in a Video Game World, In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, Boulder, CO, Sept. 3-6, 1996.
8. Pane, J.F. and Myers, B.A. (1996) Usability Issues in the Design of Novice Programming Systems, *Technical Report CMU-CS-96-132*, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. 1996.
9. Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. New York, Basic Books.
10. Pea, R.D. and Kurland, D.M. (1986) On the Cognitive Effects of Learning Computer Programming, *MIRRORS OF MINDS: Patterns of Experience in Educational Computing*, Pea and Sheingold (Eds.). Ablex Publishing Corp., Norwood, NJ.
11. Repenning, A. and Ambach, J. (1996) Tactile Programming: A Unified Manipulation Paradigm Supporting Program Comprehension, Composition and Sharing. In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, Boulder, CO, Sept. 3-6, 1996.