CREATING GAMES OR DEVELOPING PROGRAMS? DOCUMENTING THE USE OF STAGECAST CREATOR™ AS MODELING TOOL IN ELEMENTARY SCIENCE

Loucas Louca

ABSTRACT
This paper reports a descriptive case study investigating the use of a computer-based programming environment (CPE), Stagecast Creator™ (SC) for collaborative fifth grade modeling in science. SC is a non-linear, object-oriented CPE that uses visual program language (PL). The purpose of the study was to investigate and provide rich descriptions of student approaches to scientific modeling through the use of a visual PL. I analyzed student activities and conversations working with SC during an after-school club using contextual inquiry, analysis of student conversation and artifact analysis. The findings suggest that the visual CPE influences learning in a particular direction. The purpose of student work was to create representations of natural phenomena. Despite this, students in the study used SC for creating games. They focused on the overall story underlying their games which they then translated into programmable rules. This was in conflict with SC's object-oriented interface, suggesting that despite assumptions for the opposite, object-oriented interface may pose difficulties to young science learners. However, telling the story of individual causal agents supported collaborative scientific modeling among learners in the study. Programming in SC was much easier than other traditional CPEs, but reading their programs was less tangible.

KEYWORDS
Scientific modeling, object-oriented interface, computer-based programming environments

INTRODUCTION

There is a longstanding awareness of the need to help students learn with and about *models* and *modeling* in science, and computer-based programming environments (CPEs) are promising tools for this (Louca, 2004). The process of scientific modeling can be compared to the process of computer programming, and modeling can be carried out through developing a computer program. In this view, programs produce *computer microworlds* which are structured environments that learners can use to explore and manipulate a rule-generated universe, subject to particular assumptions and constraints (Pea, 1984). To represent natural phenomena through computer microworlds, students have to *deconstruct their understanding* into small programmable pieces of knowledge, in order to transform an idea in science into specific, technically precise program code. The activity of programming may also bring the constraint of *formal precision*. Students learning science often struggle with terms such as "force" or "acceleration" that have everyday, context-dependent meanings. Science students need to learn new, more refined meanings of these terms, but, as importantly (and as difficult to accomplish), they need to learn the practice of quantitative precision: For an idea to be useful in science, it should be made sufficiently precise in order to maintain consistent meaning across different contexts (Hammer & Elby, 2003). Like mathematics, programming can be a language for using in developing understanding in science (Sherin, 1996), which can also help students to develop new, refined meanings of terms (Hammer & Elby, 2003). This has been the approach of a number of studies about computer-based

modeling in science education (diSessa et al, 1991; Louca & Constantinou, 1999; Redish & Wilson, 1993; Sherin, 1996; Sherin et al, 1993; White & Frederiksen, 1998; Wilensky & Resnick, 1999).

Programming has at least two important advantages over any other written form of language such as mathematics. First, unlike mathematics, *a program can be executed* on a computer and produce observable results instantly, which provides a way of expressing scientific models that may be more tangible and accessible for young learners than any other ways. Second, *the code itself can be more easily read and explained* than algebra for instance. Rather than equations depicting relations among quantities, lines of code represent procedural instructions that, given a sufficiently accessible language, students can read and follow step by step.

This study seeks to describe in detail the ways that a group of fifth grade learners used a graphical computer-based programming environment (CPE), Stagecast Creator™ (SC), as a modeling tool for natural phenomena. In this paper I compare findings from this study with a similar one (Louca, 2005a) in which learners used a textual CPE, to highlight the possibility that SC's graphical PL pushed learning in early elementary science into a particular direction. The data analyzed here suggest that student work differs between different PLs, and those differences seem to originate from the different characteristics of the CPEs themselves. The graphical PL seemed to put students in a mode of using the software to develop representations of stories, while having more conversations, whereas data from a similar study (Louca, 2005a) suggest that a textual PL seemed to put students in a technical mode of work with limited conversations. Although programming with a graphical language seemed to be much easier than the textual one, reading code in SC was less tangible and less productive for scientific modeling.

Models and modeling in science education

Models are systematic representations of physical systems (Glynn & Duit, 1995), which include rules and relations between objects, physical values and physical concepts. They are used to describe, represent and explain the mechanisms underlying natural phenomena. Good models extend across individual systems and are descriptions of our understanding of fundamental mechanisms in nature. Models can play a dual role in science learning: they can be both tools for learning and learning outcomes; in this paper only models' role in scientific learning is studied. Science proceeds through the construction and refinement of models, and learning science should include developing understanding about natural phenomena by constructing models (Golin, 1997), as well as learning the process of developing and refining those models (National Research Council, 1990; White & Frederiksen, 1998). Scientific models and the process of scientific modeling can provide an approach that can guide the process of science learning. In this context, the development and refinement of models can achieve better outcomes than are currently possible in many educational systems (Harrison & Treagust, 1998; Bell, 1995; Grosslight et al, 1991).

Stagecast Creator and visual programming

Currently there are a number of widely-varying CPEs designed for young learners including formal programming (STELLA), textual programming (Microworlds), animated programming (ToonTalk), 3-dimensional programming (Alice, Squeak Land), visual programming (RoboLab), and graphical programming (Stagecast Creator & Icicle). Given the wide range of PL particularly developed for young learners, it is necessary to define which characteristics meet learners' programming needs and learning habits in science. Traditional studies have failed to describe in detail how learners use different CPEs; they usually study the effects of programming on skills such as problem solving among learners using pre/post tests designs or describe the characteristics and capabilities of software, (i.e., Rader et al, 1997; Smith et al, 2000), without any descriptions of how student use them. To program through typing instructions (in textual and formal programming CPEs) might sound more difficult than assigning rules to objects (in graphical CPEs), especially given the lack of any scaffolding for programming, but running a program by executing instructions is very different from executing a set of conditional rules (Ko, Aung, & Myers, 2005) . Additionally, reading programs in a textual PL may seem harder, but one may wonder whether reading graphical representations of rules is any easier.

SC (Smith & Cypher, 1999) that was used in this study has several key features. It is an object-oriented CPE that allows construction of symbolic simulations, designed to eliminate problems related with learning a new PL. Programming is done through direct manipulation of the objects, assigning them with graphical "if-then" rules (Smith & Cypher, 1999): for a given situation (condition), an action is determined. During programming the software records the user's actions, creating a script consisting of rules. Thus, there is no need of using a keyboard, although that is an option. When in "running mode," SC runs through the objects' rule lists, trying to match the world around an object to the "if" condition of one of the rules. When such a match is found, the world around the agent is changed to that represented by the rule's action ("then" action). When a rule runs, it is marked with a green light, whereas a red light indicates that a rule cannot run. Additionally, SC has a debugging tool, which in a given situation can be used to highlight the problematic part(s) of the rule(s).

Although interactions between characters are feasible, relations between different rules are very limited. By default, for each step of each object, SC starts from the first rule in the object's rule list. If that rule cannot run, SC checks the next rule in the list and so on. When specifically set, SC can run rules randomly, in sequence (first, second, third etc) and simultaneously; however any other relations between rules such as calling a rule within another, are not possible. This unique characteristic of how SC executes the rules influences the results of programs. On the other hand, unlike traditional CPEs (such as Logo (Papert, 1993)), variables are clearly differentiated from the rest of the code (they are represented with boxes named after variables), are stored "behind" each object, and can be accessed by double-clicking on the objects. Variables can be incorporated in the program by simply dragging them into the rules.

Differences between programming and pre-packaged simulations

Programming tools such as the one investigated in this study differ from pre-packaged computer simulations of natural phenomena. In pre-packaged computer simulations, students are provided with an already constructed simulation, which they can simply explore. However, learners neither deal with the development of the simulation nor with the representation of concepts/variables in the simulation. In contrast to simulations, the programming tools of the type used in this study require that *the learners participate in the development and debugging of the code to generate a simulation*. Rather than provide them with a model to explore, these tools engage students in the process of developing the models themselves. "Research has shown that the process of creating models (as opposed to simply using models built by someone else) not only fosters model-building skills but also helps develop greater understanding of the concepts embedded in the models (Colella et al, 2000, p.1)."

METHODOLOGY

Study setting

This study took place at a suburban elementary school in Maryland, USA, during September – December 2002. I set up an afternoon computer/science clubs for fifth graders, meeting once a week for 90 minutes. The school administration selected fifteen fifth graders among those who volunteered for the study, to be representative of the school population: African-American, Latino, Asian and Caucasian students; a third of them were girls. During the study, they worked in small groups of 2-3 members each. The distribution of students in the groups reflected both cultural and gender diversity among the groups, and the effort was to have group members who could work collaboratively and communicate successfully. Participants were of mixed ability levels. They also had some experience with computers through regular visits to the school's computer lab, even though none of the participants had previously used either software that was used in the study.

Study phases

The study was divided into two phases. Phase I took place during the first 6 meetings and was mainly devoted to learning the PL and becoming familiar with the CPEs. Phase II was devoted to exploring ideas in science with the use of the CPEs. The findings reported in this paper are based on the data collected during phase II. During phase I, the teaching focus was on the process of rule creation.

Students familiarized themselves with the environment using the software tutorial that was previously found to be successful (Louca et al, 2003). Additionally, I introduced students to several pre-programmed simulations and asked them to make changes in them. This approach can help students to focus on the process of developing and editing rules and on their role in creating a simulation. More details about phase I can be found in (Louca, 2004). During phase II, each group developed a representation of a natural phenomenon. Prior to any work, group members collaboratively decided the topic of their final project, thus enabling and providing support for different student preferences and likes. Each group selected a different phenomenon.

Data sources

Two primary data sources were used in this study and are reported in this paper: (1) *audiotaped group work with SC,* and (2) s*tudent programs*. During weekly meetings, all small group discussions during computer work were audiotaped. Additionally, two groups from each team were videotaped, and the videos were used to develop detailed video cases of student work with computers (Louca, 2004). Student programs were also collected on a weekly basis for analysis. I also used two secondary sources of data: (1) screen-captured video from computer screens and (2) my weekly journals before and after each meeting. Secondary data sources were used for guidance during analysis to support analysis and findings.

Data analysis

This is an interpretive qualitative study (Bogdan & Bilken, 1998) following the tradition of case study (Merriam, 1988; Stake, 2000) seeking to describe fifth grade student work with SC while developing models of natural phenomena. For analysis purposes, the whole group of students was treated as a single Case, and individual group work as case study units. Data analysis and presentation of findings were based on all groups from the club (case study units) following their work in detail for almost two months of meetings (case study sub-units). For the data analysis I used three different methods: (1) *contextual inquiry*, (2) *analysis of student conversation*, and (3) *artifact analysis* of student programs. Below I provide descriptions of the 3 types of analysis.

Contextual inquiry

I analyzed transcribed conversations using a modified version of *Contextual Inquiry* (Druin, et al 1999), a method of collecting and analyzing data of children's activities and conversations in a particular context. Transcripts of audiotaped conversations were separately coded for *activity and conversation patterns*. Each utterance was coded separately (decision adopted from a student-interaction study (Underwood et al, 1996). Moments of silence were represented in separate cells. A total of 36 codes emerged from the data, following open coding from grounded research (Strauss & Corbin, 1998) - for a summary of the codes see Louca (2004). After the list of codes was finalized, I reviewed once more all the coded transcripts, checking for consistency in the applied coding. Coding was then repeated by another coder, who used the same final list of codes. Differences in codes applied were resolved through discussion.

Student *activity and conversation patterns* were separately presented in time-line graphs (technique adopted from Schoenfeld (1989)). For each student group two graphs were produced; one for the activity and the other for the conversational data. Codes were used on the y-axis of the graphs and the number of utterance on the x-axis; examples of these graphs can be found in figures 1, 2 & 3. Graphs from units of the Case were compared to isolate similarities in the combinations of the patterns of student activities and conversations. Similarities were grouped together, forming *activity and conversation types* which refer to specific combinations of student work or conversations. Only activity or conversation types that were observed in all groups' data are reported in this paper. These types of activities or conversations also emerged from the data and are descriptive of student work or conversations.

Analysis of student conversation
Analysis of student conversation is a multidisciplinary approach to analyzing transcripts of student conversations. It uses research techniques and approaches originating from linguistics, educational psychology and educational research (Edwards & Mercer, 1995), and use transcripts as gateways to student thinking and experience. This approach has been used by a number of researchers who analyzed student conversations in science and mathematics (e.g. Ball, 1993; Gallas, 1995). Analysis of student conversation uses discourse (Sinclair & Coulthard, 1975) to describe the content of the conversation and the context in which the conversation takes place (Edwards & Mercer, 1995), for which contextual inquiry cannot account.

Patterns of student conversations (revealed by contextual inquiry) and analysis of student conversation are presented together to triangulate findings (Stake, 2000). Triangulation helps to support claims of student work and conversations, by using two different analyses from two different theoretical perspectives to point to similar findings. This combination of findings provides a better, more detailed picture. Snippets of student work and conversation that are presented in this paper were selected as examples to support the claims from this study. Combinations of contextual inquiry and analysis of conversation were used to develop several case studies of the two SC groups that were videotaped. That work is presented in detail elsewhere (Louca, 2004).

Artifact analysis of the designed microworlds
For the purposes of the artifact analysis, I reviewed student programs. During phase II, each group developed several models (programs) of a particular phenomenon per group, each revised from a previous one. Artifact analysis focused on the differences and the complexity of those programs and on the particular ways that students represented phenomena through computer programs. My goal was to map possible relationships between the types and the characteristics of students' scientific models and the findings from both contextual inquiry and analysis of student conversations. I also investigated the kinds of representations that students used focusing on whether programs represented causal relationships between physical values.

FINDINGS

In this section I provide descriptions of student approaches to planning, creating and debugging rules, and how they used their programs to represent natural phenomena. Space precludes numerous and lengthy examples of student work, and for this reason I provide small excerpts from the work of two groups of students. Findings that are presented below were confirmed from all groups of the study. Additionally, findings discussed below are limited to descriptions of student scientific modeling with SC – more details about the ways that students used SC can be found elsewhere (Louca, 2005a) as well as comprehensive comparison of modeling with SC and other traditional CPEs (Louca, 2005b).

Approaches to planning
Students in the study planned their work by *talking about the scenario they were about to program*. Although the only requirement for their work was to develop representations of natural phenomena, all groups decided to develop games that would include those representations. For this reason, students tended to focus on the overall story underlying their games and talked in detail about how their simulation would look. In one group for instance, Annie and Bryan described to create a game that would be about shooting down helium balloons that travel at different speeds. They talked about how balloons would be shot down and about scoring, providing lots of details about their game scenario. Their conversation was about a series of events that students would show through their simulation. This is reflected in the following excerpt.

> *Annie*: we, the name that we end up deciding to call our game balloon shoot-out and the idea is that are series of colored balloons and the different colors make them bigger. And…
> *Bryan*: for example like the less point there, the bigger balloons are.
> *Annie*: yea, like gold is to be tiny but it's worth 50 points. And there's different ones with <inaudible> but gray we got a gray balloon and it'll be a wipe-out and like the points will be gone. And through deciding may,

we don't know how many levels we're gonna make, and we decided that 3 wipe-outs is the end of that level. And if you are like level 2, you have to go down to go to the bottom.

Students broke down their ideas based on the sequence of the events in their story: what would happen first, second, third and so on, similar to describing scenes from a movie. During that time, the use of the PL was minimal. As Figure 1 shows, reference to PL was much less than reference to the underlying story of their designs. It is possible to suggest that students were in the "mode" of deciding their scenario and they were overwhelmed by the details they wanted to include.

Approaches to creating and debugging rules

During programming, students maintained their focus on the succession of events in the underlying story of their designs, and their work and conversations reflected an effort to *translate the details of their scenario into programmable rules*. For instance, Annie and Bryan talked about specific details of their game (e.g., have balloons moving with different speeds) and how to program them. Programming with SC was an interactive process of work and conversations. Figure 2 presents student conversation patterns during programming and debugging with SC. In a collaborative effort, students translated the details of their story into rules, as it is represented in the "tell story with no code" and "how to program" coding categories, which show that their focus *was on creating a simulation that would show their story*, as a succession of events. This is also reflected in the following excerpt.

Annie: yea, but we want different colors [for the balloons].
Bryan: can we make up a green.
Annie: and gray and yellow and gold.
Bryan: the slowest is red, right?
Annie: yes. The slowest is red.
<silence while clicking/typing/looking on the screen>
Annie: all right. Let's make it move, um 2 boxes, at a time
Bryan: like, why not 1?
Annie: cause that would be so easy to shoot.
<silence while clicking/typing/looking on the screen>
Annie: done. Now let's play. Right.
<silence while clicking/typing/looking on the screen>
Bryan: No, that should be the gold.
Annie: yea. That's too fast.

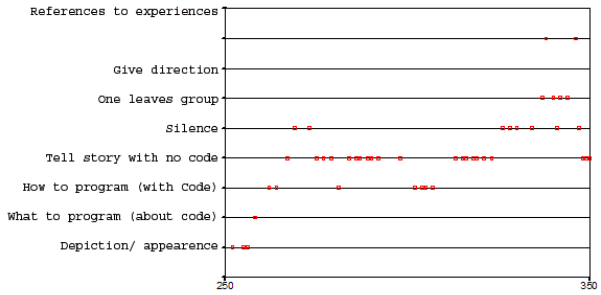

Figure 1. Conversation during planning

Early programs consisted of a number of rules representing successive "scenes" from their story, which SC ran in sequence. Students wrote each rule and tried it immediately before moving to the next one. If a rule did not work, students deleted it and created a new one, without spending any time figuring out what was wrong (debugging). Figure 3 illustrates that pattern. There is no conclusive evidence, however, about whether students deleted a rule only to recreate the same one or whether they knew specifically what to do differently in the new rule. There is evidence, however, that in most of the cases students did not try to see what was wrong with the old rule, despite the available debugging tools in SC. At least in a few cases, however, students talked about what was wrong with their programs and, instead of making changes in their rules, they deleted them and created new ones.

Debugging was only in the form of deleting rules that had unwanted effects or changing the rule sequence. Syntactical bugs are almost impossible in SC because the software provides a lot of scaffolding for creating rules. The user has simply to demonstrate to the system the desired behavior or fill in blanks about the conditions of each rule and its action.

In this phase of their work, due to their focus on programming, the use of PL was much higher than before. Student conversations were mostly coded as exchanges of talking with the PL and talking about

their scenario, reflecting an effort to translate their ideas into rules/code to program. Although the overall scenario was still their concern, their conversations seemed to become more technical and focused on the kinds of rules that they would create. As a communication medium, PL can be more precise than usual, everyday conversations because it is used to develop instructions for particular object behaviors to follow in the program. In this sense, a conversation about a disagreement about the motion of different balloons may better be supported by the use of the PL, which can provide students with clear, precise and tangible ways of talking about their ideas.

Additionally, as Figure 2 illustrates, students started making references to everyday experiences in order to convince each other about how to program specific aspects of their games. Interestingly this mostly happened in the context of debating about particular details of the rules (such as how much should the speed of a falling ball change over time). This need for supporting their program ideas arose when students started debating the details of their rules and to do so, students turned into everyday experiences. This might have been the beginning of conversations about the mechanism that caused the phenomena under study. Differences in the behavior of objects, discussed in the context of talking about particular programming ideas, can lead to conversations about what actually is causing the phenomenon.

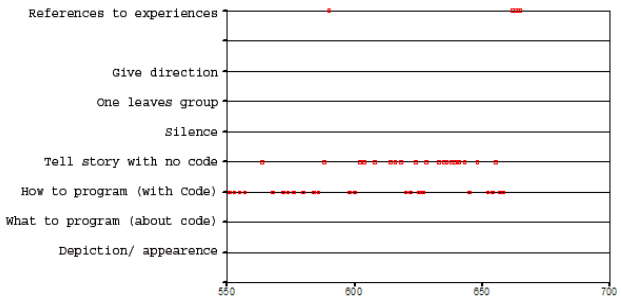

Figure 2. Conversation during programming and debugging

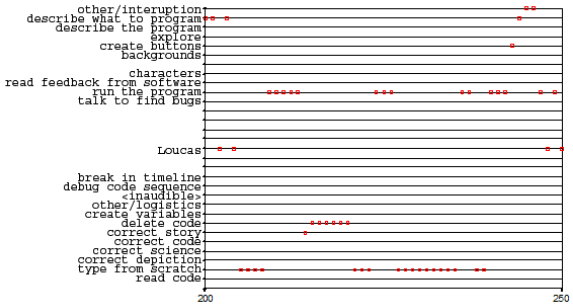

Figure 3. Activity pattern during programming and debugging

Approaches to using rules (code) as a representation of a phenomenon

To talk about causal mechanisms, the students had to shift their focus from telling a story with the simulation, to talking about the different variables (agents) in the story that affected object behavior such as food, energy, speed, acceleration. The students had conversations about how things happened in their simulation, utilizing the scaffolding of programming of SC to talk about the causal mechanism of the phenomenon. While working on their balloon game, for instance, Annie and Bryan entered a discussion about representing a mechanism that would cause their balloons to move according to the amount of helium inside them. Their rules thus far were simply descriptive of balloon behavior, but they soon realized that they could develop a set of rules that could actually "read" the amount of helium (variable) in each balloon and cause it to move accordingly. This would lead them to a more general mechanism that could be applied consistently for all balloons with different amounts of helium and "create" a simulation, instead of simply replaying a pre-sequenced series of rules.

Scaffolding that SC provides for rule creation seemed to support conversations about the mechanism; the difficulty, however, was to start such a conversation. The students worked towards developing games that would include natural phenomena, rather than representing the phenomena *per se*. They focused on the overall story and the details of their games, which seemed to get in the way of scientific modeling. This could have been partly because a modeling conversation would require students to talk about what caused the changes in the object's behaviors in their game scenario. Creating multiple rules for multiple behaviors is much easier than creating fewer, more general rules that *cause* those different behaviors and the changes in those behaviors over time.

The following excerpt, during which Zen and Sean developed a model for a falling ball, highlights how students could move from a simple descriptive model to a revised causal one. To do that, as the excerpt

illustrates, the students had to use rules as the representation of the phenomenon, rather than as the way to create a simulation that would show the phenomenon. Reading their programs seemed to be a context that afforded productive conversations about causality in nature, which is the heart of scientific modeling.

Part 1: Creating a descriptive model
As part of their program, Zen and Seth had to program a ball falling towards the ground. Zen suggested to approach this situation by "marking each vertical point of the ball's motion with its own speed."

> **Zen:** … When the vertical [position of the ball] is for instance 34 then it'll go 1 square, when it is 29 it will go 2 squares and when it's 19 it'll go 3 squares. When it's 9 it'll go 4 squares. So faster and faster and faster.
> **Sean:** it's the same as Bryan's idea, except I'm, we're just making it so that there's a certain time to use the rules where see how it's vertical now, just move the ball down…
> **Zen:** every, every like 9 squares it will go, it'll skip another, it will skip another <inaudible>every 9 vertical numbers it going down, it'll skip a square.

Zen and Sean developed a program which included a separate rule for each vertical position of the ball's motion, assigning each position with a particular speed for the ball (see Figure 4). This resulted in a number of rules, each associated with a particular position of the ball and with a particular speed. Up to that point their intention was to recreate the ball's motion on the computer screen. They knew that the ball changes speed while moving, even though they did not specifically show how that happened in their program.

Although their first program was not representing anything about how the phenomenon was caused, there was a mechanism (although possibly not scientifically accurate) underlying their program as a hidden assumption (Hammer, personal communication): in each subsequent rule, students added one grid square to the distance that the ball moved. Their program represented the phenomenon in small pieces, each piece (rule) associating the ball's position with its speed. That succession of rules represented the changes in the motion of the ball, without however, any reference to how these changes were caused. What was missing from their model was a representation of the relation between the different snapshots (rules), which is the rate of the change of the speed (acceleration). Because of the absence of the representation of that relation in their program, one may argue that this was not a model, but rather a simple depiction of the phenomenon made up of different scenes that one follows another. At the same time, however, this program was a great opportunity for students to discover the need for inventing a concept such as speed or velocity and using it in the program.

Part 2: Refining their initial model: working towards representing causality
Sean and Zen's initial model was specifically written to work in one particular situation, only when the ball was released from a height of 30 grid squares. This was a problem that they soon encountered. Their program did not work when the ball was let fall from 28 or 31 for instance, because there was no rule for motion at that vertical point, posing the constraint of maintaining applicability of their program (model) at different heights. Similarly, if the ball started falling form vertical 27, it would start with a speed of 3 grid squares per clock tick.

> **Zen:** this is the [rule for the vertical position] 29. So [talking to Sean who has the mouse] click on this, click here, and then go here and 1 down. Thank you very much! Now this will work. This will go second from last. So it goes here…
> <silence while clicking/typing/looking on the screen>
> **Zen:** yes. Thank you. Now watch.
> <silence while clicking/typing/looking on the screen>
> **Zen:** o, oh!
> **Sean:** it's stopping at 28!
> **Zen:** ok. Please! What's wrong with it?
> **Loucas**: let's think about it a little. Fro which vertical do you have rule?

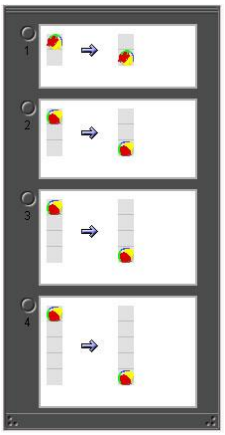

Figure 4. Zen and Seth's first program for the falling ball

Zen: here is for 30, this is for 29, 27, 24, and 20.
Sean: no it goes 1 when it's 30.
Zen: yea, exactly, it falls 2 when it reaches 29.
Loucas: ok. What would happen if you let the ball fall from 31?
Sean: it will not work!

This led to a conversation that I helped initiate about what is going on in the phenomenon. During that conversation, Zen indicated that what was changing in every rule was how many squares would the ball move. The relation between their rules (in every subsequent rule, the ball was moving an additional square) made Zen start thinking about the idea of having one rule that would do just this.

Zen: so how can we make the program that no matter where you let the ball go, it works?
Sean: have rules for every vertical!
Loucas: right, but then how much are they going to move? Like, if I let it go from 30, at 29 is going to move, how much?
Zen: 1
Loucas: and if I let it go from 31…
Zen: then it's probably going to have a different speed…..
Loucas: so what is changing in your program?
Zen: what do you mean?
Loucas: what is different between this rule and this rule?
Sean: the vertical
Zen: and how much moves
Loucas: right, so what I see your rules doing is in every next vertical, you are adding 1 to how many squares your ball moves. Here you have one, then 2, then 3.
Zen: oh, for every time it falls it gains 1. I should just have done that! So have a rule that every time it falls it gains 1. I was thinking about that but then I didn't know how.

At that point Zen started talking about the mechanism of the phenomenon, in the context of writing a rule that could simply add 1 to the number of squares that the ball would move next. In this sense, instead of having multiple rules that SC would simply run sequentially, students could have one rule that could create the phenomenon.

My claim here is not that students could make such moves by themselves (even though it is not unlikely). Rather, I highlight the role of reading their program consisting of a number of rules, in getting students thinking about the mechanism that caused the phenomenon. More importantly, reading their rules and looking at what each subsequent rule did, helped Zen realize that they could replace them with one rule that would increase the number of grid squares that the ball would move. This rule had three advantages over their previous program. First, a single, more "general" rule could account for and create the whole motion of the falling ball, providing a clearer representation of the phenomenon's underlying mechanism. Second, the refined program could also be used with balls that were let to fall from any height, providing more applicability power to their model. Thirdly, their revised program represented the mechanism that was responsible for the changes in the behavior of the falling ball (acceleration) and as such it was representing in code (not just in the resulted simulation) the actual phenomenon. Students had made a clear move towards scientific modeling.

DISCUSSION

In this paper I have provided descriptions of the different ways that students in the study used SC in the context of scientific modeling. At the outset of their work, students focused on the overall story of their games and talked about the different scenes. During programming, they maintained a focus on the overall story, translating the scenes of their scenario into rules which was accompanied by extensive conversations. Even though students seemed to be able to use code as a representation of the causal mechanisms of natural phenomena, the focus on the overall story seemed to get in the way of modeling.

In a different study reported elsewhere (Louca, 2005a) students worked in a similar learning context with a textual CPE (Microworlds (MW)). In that study, MW students made plans about the structure of their programs according to the parts of the phenomena they were modeling, whereas SC students broke their programs in sequential scenes. Additionally, during programming MW student shifted their focus towards creating a simulation that would execute and looked satisfactory, whereas SC students maintained a "story-telling" mode of work. At the same time MW student conversations were "technical" and much more limited than SC student conversations.

Comparison of the findings from the two studies, even though is not meant to generalized findings, suggests the possibility that the differences in the PL and other characteristics of the CPEs seemed to push learning into distinctive learning directions. Overall, in the context of preparing to write a program to represent a phenomenon, MW seemed more likely to trigger modeling conversations and practices than SC. This does not mean that SC was non-productive for scientific modeling; rather, the way that students were *inclined* to use it early in their work was not supportive of ways that could trigger conversations, activities and thinking for modeling in science. Creating games, even through much easier programming, was steering students into a direction of creating "quality" games but not necessarily "quality" models of natural phenomena. Although the process of formal programming that MW seemed to push students into was more difficult, the program representation was much more meaningful for students and reading their programs much more tangible. Programming in SC was more tangible, but program representation was more abstract.

Write a program vs. create a game

The comparison between findings of this study and the one where students used a textual PL (Louca, 2005a) suggests that the students conceptualized their work with CPEs differently. Students used SC to *create games*. They talked about a game scenario, what their game would include, how scoring would be done and what the player was supposed to do. Students broke down their programs based on the ideas they wanted to show in those scenes. They seemed to have "quality" criteria related to the characteristics of computer games, including depiction, scoring rules, and "a purpose for their game." Their program decisions were based on features of their games, and how to create rules that would cause objects' behaviors that fit in their game scenario. Early programs included a number of independent rules, each corresponding to a "scene" of their game.

On the other hand, MW students worked mostly through a mode of *formal programming* and their conversations were mostly *technical* (Louca, 2005a). They talked about the structure of their programs, the number of subroutines in their programs, the code they would use and how all that would fit with a proposed simulation, without any reference to details. They focused on writing programs without bugs and they broke down their programs in segments that were meaningful for programming.

Formal programming and creating games can both have disadvantages and advantages. For MW, students had to spend a significant portion of time doing "lonely," technical work (typing code and debugging), but as soon as they had a program that worked, they could focus on the phenomenon represented. For SC, due to the students' focus on the overall story, "game" and "science" were sometimes in conflict. Of course, there were cases in which a "story-telling approach" was productive for telling a story of a "causal agent" such as acceleration, gravity etc. This is similar to developing a program that creates a simulation rather than simply describes it.

Object-oriented vs. procedural programming

One of the differences of CPEs designed for young learners is the type of programming that they use. MW is a procedural, linear CPE that uses textual PL (Papert, 1993). SC is an object-oriented, non-linear CPE, which uses visual if-then rules (Smith & Cypher, 1999). Programming-by-demonstration in SC enables students to "program" a cartoon-like simulation with no awareness that they are programming (Rader et al, 1997), and thus students do not have to be concerned with the details of formal programming. SC scaffolding for programming (Smith et al, 2000) makes the process much easier. In

MW, students must use a symbolic, much more difficult language to write instructions for the different objects.

Despite the object-oriented interface, SC students planned their work in a non-object oriented manner. They thought about the system, its behaviors and characteristics, but not about the system's individual objects and their behaviors. This added the difficulty of abstracting the desired system actions and translating them into specific rules assigned to characters, which might have been one of the reasons that SC students had extensive conversations during programming. The actual programming was much more tangible than MW because students assigned behaviors to characters, but was in conflict with their plans. For MW students, however, planning and programming was done on the same basis, and was much more straight-forward, despite the added difficulty of formal program writing (Louca, 2005a).

Assigning behaviors vs. giving instructions

Another key difference between SC and MW is the way that programming is done. Programming in MW consists of typing instructions for objects to follow (Papert, 1993), whereas programming in SC is done by assigning behaviors to objects (Smith & Cypher, 1999). Due to the way programming is done in SC, students in this study usually started with creating sequences of independent components of the phenomena, whereas students working with MW (Louca, 2005a) seemed to divide their programs into pieces based on visual purposes of the simulation. In both cases, their first programs were simply descriptive of the natural phenomena. Once both groups had descriptive models, it seemed easier for SC students to make the move towards a single rule that would create the different behaviors instead of having multiple rules, one for each different behavior, which was a step towards creating a causal model. Of course, this happened only when students focused on the behavior of a character and not on the behavior of the system. It seemed to be harder for MW students to proceed towards creating a causal model, possibly because of the lack of any scaffolding from the software.

There was, however, another difficulty that SC students encountered. SC makes simple programming easier for students, but complex programming harder (Smith et al, 2000). When students started thinking for ways to include causal relationships between snapshots of their programs, they shortly found out that a single rule cannot include multiple different behaviors and relations between different rules, whereas there was not such a limitation in MW. For this reason, revisions of their programs resulted much more advanced programs in MW towards the direction of developing causal models.

Easier programming vs. easier program representation

Program representation in SC is more abstract, which seemed to make it harder for students to read and make revisions to their programs. Program representation in SC is done by graphical representations of the rules, which serve only as simple reminders of the rules. To read a program, students in SC had to figure out what each rule was doing by interpreting graphical reminders of the actions of each rule. On the other hand, program representation in MW seemed much more meaningful and straight-forward, because students could simply read the code to understand the program.

CONCLUSION

In this study I have described in detail the use of SC for collaborative fifth grade modeling in science Findings suggest that the characteristics of CPEs influence the "mode of work" that learners enter when using them, pushing their learning experiences into different directions. Findings reported here could serve as a basis for further investigation into how learners use programming as a modeling tool in science. They can help re-define the questions that teachers ask when using CPEs with young learners, and also provide teachers with a framework of what to look for during modeling with CPEs in science. It was evident that students in this study entered a particular "mode of work," which seemed to depend on the software they used. Therefore, teaching purposes, student abilities and learning styles should be viewed through this lens of CPEs characteristics. Lastly, findings suggest that SC has features that were productive and helpful for students in particular contexts of their work, which I highlighted in this paper. These include the context of debating difference in the behaviors represented in rules and the

context of reading their programs. Teachers that use computer-based modeling in their class may find the descriptions of those contexts useful.

While the importance of the teaching-through-modeling approach is well accepted and documented in the science education research, it is important to raise questions about how exactly modeling with young learners and CPEs look like, and which contexts and software characteristics support collaborative scientific modeling.

REFERENCES

Ball, L., D. (1993). With an eye on the mathematical horizon: dilemmas of teaching elementary school mathematics. The elementary school journal, 93 (4), 373-397.

Bell, Ph. (1995, April). How far does light go? : Individual and collaborative sense-making of scientific evidence. *AERA*, p. 1-36.

Bogdan, R., C. & Bilken,S., K. (1998). *Qualitative research for education: An introduction to theory and* methods. MA: Allyn & Bacon.

Colella, V., A., Klopfer, E., & Resnick, M. (2000). Adventures in modeling: Exploring Complex, Dynamic systems with Starlogo. NY: Teachers College Press.

diSessa, A. A., Abelson, H., & Ploger, D. (1991). An overview of Boxer. Journal of Mathematical Behavior, 10, 3-15.

Druin, A., Bederson, B., Boltman, A., Miura, A., Knotts-Callahn, D. & Plat, M. (1999). Children as our technology design partners. In A. Druin (Ed.). The Design of Children's Technology. San Francisco: Morgan Kaufmann Publishers, Inc.

Edwards, D. & Mercer, N. (1995). Common knowledge: The development of understanding in the classroom. NY: Routtedge.

Gallas, K. (1995). Talking their way into science: hearing children's questions and theories, responding with curricula. NY: Teachers College Press.

Glynn, S. M. and Duit, R. (1995) Learning science meaningfully: Constructing conceptual models. In S., M. Glynn & R. Duit (Eds.), *Learning science in the schools: Research reforming practice.* NJ: Lawrence Erlbaum Associates.

Golin, G. (1997). Structure of scientific knowledge and curriculum design. *Interchange*, 28 (2,3), 159-169.

Grosslight, L., Unger, Chr., Jay, E. and Smith, C., L. (1991) Understanding models and their use in science: Conceptions of middle and high school students and experts. *Journal of Research in Science Teaching*, 28 (9), 799-822.

Hammer, D. M., & Elby, A. (2003). Tapping epistemological resources for learning physics. Journal of the Learning Sciences, 12(1), 53-90.

Harrison, A., G. and Treagust, D., F. (1998) Modeling in science lessons: Are there better ways to learn with models? *School Science and Mathematics*, 98 (8), 420-429.

K

o, A. J., Aung, H. H., and Myers, B. A. (2005). Design Requirements for More Flexible Structured Editors from a Study of Programmers' Text Editing. In the Proceedings of the Computer-Human Interaction Conference 2005 (CHI 2005), Oregon USA.

Louca, L (2005a). The Syntax or the Story Behind it? A Usability Study of Student Work With Computer-Based Programming Environments in Elementary Science. In the Proceedings of the Computer-Human Interaction Conference 2005 (CHI 2005), Oregon USA.

Louca, L. (2005b). Modeling through programming in early science education: A comparative description of scientific modeling with Microworlds Logo™ and Stagecast Creator™. Paper presented at the European Science Education Research Association 2005 Conference (ESERA 2005), Barcelona, Spain.

Louca, L. & Constantinou, C. (1999, June). The use of Stagecast Creator in constructing modeling skills in physical science: The case of the single lens camera. Proceedings of the Forth International Conference on Computer-Based Learning in Science, University of Twente, Enschede, The Netherlands.

Louca, L. (2004). Case studies of fifth-grade student modeling in science through programming: comparison of modeling practices and conversations. Unpublished doctoral dissertation, University of Maryland, College Park.

Louca, L., Druin, A., Hammer, D., & Dreher, D. (2003). Students' collaborative use of computer-based programming tools in science: A Descriptive Study. In B. Wasson, St. Ludvigsen, & Ul. Hoppe (Eds.). Designing for Change in Networked Learning Environments: Proceedings of the International Conference on Computer Support for Collaborative Learning 2003 (CSCL) (pp. 109-118) The Netherlands: Kluwer Academic Publishers.

Merriam, B. S. (1988). Case studies research in education. A qualitative approach. San Francisco, CA: Jossey-Bass, Inc., Publishers.

National Research Council. (1990). *National Science Education Standards*. Washington, DC: National Academy.

Papert, S. (1993). The Children's machine: Rethinking school in the age of the computer. NY: Basic Books.

Pea, R. (1984). Intergrading human and computer intelligence. Technical Report no. 32. Banks Street College of Education: New York, NY.

Rader, C., Brand, C., & Lewis, Cl. (1997). Degrees of comprehension: children's understanding of visual programming environment, Communications of the ACM, 351-358.

Redish, E. F. & Wilson, J. M. (1993). Student programming in the introductory physics course: M.U.P.P.E.T. American Journal of Physics, 61 (3), 222-232.

Schoenfeld, A. H. (1989). Teaching mathematical thinking and problem solving. In L. B. Resnick & B. L. Klopfer (Eds.), Towards the thinking curriculum: Current cognitive research (pp. 83-103). Washington DC: ASCD.

Sherin, Br. (1996). The Symbolic Basis of Physical Intuition. A Study of Two Symbol Systems in Physics Instruction. Unpublished dissertation Thesis.

Sherin, Br., diSessa, A. A., & Hammer, D. (1993). Dynaturtle revisited: Learning physics through collaborative design of a computer model. Interactive Learning Environments, 3 (2), 91-118.

Sinclair J. McH., & Coulthard, R., M. (1975). Towards an analysis of discourse: The English used by teachers and pupils. London: Oxford University Press.

Smith, D., C. & Cypher, Al. (1999). Making programming easier for children. In A. Druin (Ed.). The Design of Children's Technology. San Francisco: Morgan Kaufmann Publishers, Inc.

Smith, D., C., Cypher, A., & Telser, L. (2000). Novice programming comes of Age. *Communications of the ACM*, 43 (3), 75-81.

Stake, R. E. (2000) Case Studies. In N. K. Denzin & Y. S. Lincoln (Eds.), Handbook of qualitative research. (435-454). Thousand Oaks, CA: Sage.

Strauss, A., & Corbin, J. (1998). Basics of qualitative research. Techniques and procedures for developing grounded theory. Thousand Oaks, CA: SAGE Publications.
Underwood, G., Underwood, J., Pheasey, K., & Gilmore, D. (1996). Collaboration and discourse while programming the KidSim microworld simulation. Computers & Education, 26, 143-151.

White, B. Y. and Frederiksen, J. R. (1998). Inquiry, modeling and metacognition: Making science accessible to all students. Cognition & Instruction, 16 (11), 3-118.

Wilensky, Ur., & Resnick, M. (1999). Thinking in Levels: A Dynamic Systems Approach to Making Sense of the World. Journal of Science Education and Technology, 8 (1), 3-19.

Loucas Louca
Post-doctoral Research Associate,
Learning in Science Group,
University of Cyprus,
No 101, 19 Leda Street, Nicosia 2015, Cyprus.
Email: Louca.l@cytanet.com