ABSTRACT

Title of dissertation: CASE STUDIES OF FIFTH-GRADE STUDENT MODELING IN SCIENCE THROUGH PROGRAMMING: COMPARISON OF MODELING PRACTICES AND CONVERSATIONS

Loucas Louca, Doctor of Philosophy, 2004

Dissertation directed by: Associate Professor David Hammer.
Department of Physics and Department of Curriculum & Instruction.

This is a descriptive case study investigating the use of two computer-based programming environments (CPEs), MicroWorlds^{TM} (MW) and Stagecast Creator^{TM} (SC), as modeling tools for collaborative fifth grade science learning. In this study I investigated how CPEs might support fifth grade student work and inquiry in science.

There is a longstanding awareness of the need to help students learn about *models* and *modeling* in science, and CPEs are promising tools for this. A computer program can *be* a model of a physical system, and modeling through programming may make the process more tangible: Programming involves making decisions and assumptions; the code is used to express ideas; running the program shows the implications of those ideas.

In this study I have analyzed and compared students' activities and conversations in two after-school clubs, one working with MW and the other with SC. The findings confirm the promise of CPEs as tools for teaching practices of modeling and science, and they suggest advantages and disadvantages to that purpose of particular aspects of CPE designs.

MW is an open-ended, textual CPE that uses procedural programming. MW students focused on breaking down phenomena into small programmable pieces, which is useful for scientific modeling. Developing their programs, the students focused on writing, testing and debugging code, which are also useful for scientific modeling. SC is a non-linear, object-oriented CPE that uses visual program language. SC students saw their work as creating games. They were focused on the overall story which they then translated it into SC rules, which was in conflict with SC's object-oriented interface. However, telling the story of individual causal agents was useful for scientific modeling. Programming in SC was easier, whereas reading code in MW was more tangible. The latter helped MW students to use the code as the representation of the phenomenon rather than merely as a tool for creating a simulation.

The analyses also pointed to three emerging "frames" that describe student's work focus, based on their goals, strategies, and criteria for success. Emerging "frames" are the programming, the visualization, and the modeling frame. One way to understand the respective advantages and disadvantages of the two CPEs is with respect to which frames they engendered in students.

CASE STUDIES OF FIFTH-GRADE STUDENT MODELING IN SCIENCE THROUGH PROGRAMMING: COMPARISON OF MODELING PRACTICES AND CONVERSATIONS

by

Loucas Louca

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2004

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CPEs:	Computer-based programing media
MW:	Microworlds Logo
SC:	Stagecast Creator

1. INTRODUCTION

1.1. Overview

A physical system can be modeled by identifying the objects (or the variables) that characterize the system, the functions of each object and the relationships among these objects (Krajcik, Soloway, Blumenfeld & Mary, 1998). Science proceeds through the construction and refinement of models, and learning science entails learning how this happens, in addition to learning particular models scientists have constructed (Bell, 1995; Constantinou, 1996; Golin, 1997). That is, learning in science largely entails learning the process of developing and refining models (National Research Council, 1990; White and Frederiksen, 1998).

Understanding how to help students accomplish learning about the process of developing models has been one of the central challenges in science education, and a difficult one because learners often view science as a body of factual knowledge and learning science as a matter of receiving information (Hammer, 1994). The challenge is to develop understanding about the modeling process itself as a learning tool as well as accomplishing learning through the modeling process. Research has shown (Louca & Constantinou, 2002) that learning about models and modeling can be accomplished in early middle school ages by guiding students through a process of developing and refining models about natural phenomena.

Various studies (e.g. Louca, Hammer & Bell, 2002; Samarapungavan, 1992) suggest that children do have and can deploy intellectual resources for scientific thinking, in order, for example, to coordinate different perspectives in science in the light of new evidence and experiences. Students can be observed using these thinking resources in group settings while they have conversations about physical phenomena (whereas individual formal interviews seem not to tap into these resources (e.g. Kuhn, 1989)). Data from children's conversations in science can be used to refine our understanding about their ways of learning for, about and with models and modeling in science.

In this chapter, I start by stating the research question that guided this study, which focused on how might computer-based programming environments support fifth grade student inquiry in science. Then I provide a theoretical framework to situate the study in the research of science education (models and modeling in science), technology (views about computer programming tools as means for developing models and modeling skills), and psychology (views about students abilities for scientific inquiry).

1.2. Topic and purpose

The purpose of this case study is to describe and analyze the use of computer-based programming environments (CPEs) by 18 fifth graders as a tool for developing models of natural phenomena. I investigated the use of two different CPEs, in order to develop descriptions of the students' use of these tools. These descriptions include cases of different ways that these fifth graders used CPEs when developing models in science, and cases where these students used the two different CPEs to communicate their ideas in science. Using descriptions of the different ways that these students used CPEs, I provide descriptions of students' activity and conversation patterns that are supported – or not – by the different CPEs. Based on the students' use of the CPEs, I describe the characteristics of CPEs that support student inquiry in science.

1.3. Research questions

The central research question for this study is "*how might computer-based programming environments support fifth grade student inquiry in science*?" To provide a collection of descriptive cases that seek to answer the central research question for this study, the following subsidiary questions were investigated.

1.3.1. How do fifth graders use CPEs in learning science?

In investigating the (potential) role of CPES in supporting thinking and learning in science at the elementary level, I provide detailed descriptions of how fifth graders use features of CPEs to learn in science. Given the specific characteristics of different types of CPEs, it is possible that different programming environments promote/support different ways of learning science and as such, provide unique learning environments for students. I describe the ways that students use CPEs including the process of programming for developing representations of natural phenomena.

1.3.2. What are the characteristics of student thinking in science that are supported by CPEs?

In this study I investigated student inquiry that takes place in the context of using CPEs for developing models in science. Given the idea that the use of prior knowledge, experiences and thinking strategies can be context dependant (as I justify in the theoretical framework of this proposal), and given the particular characteristics of CPEs, I provide descriptions of student thinking while working with CPEs.

1.3.3. What are the characteristics of CPEs that support collaborative modeling practices among fifth grade student in science?

My third goal in this study was to provide descriptions of two different CPEs, including descriptions of the characteristics of the process of programming in these environments. In the theoretical framework of this report I provide descriptions of several characteristics of different CPEs. In most of these cases, these characteristics have not been explicitly investigated with users to determine their effectiveness for young learners. In this study, I seek to provide descriptions of the characteristics of CPEs that support collaborative modeling practices among fifth graders, in the context of developing models in science.

1.4. Theoretical framework

In the theoretical framework of the study, I provide a discussion about models in general, specifically distinguishing the differences of conceptual and mental models. I then provide some discussion about the use of modeling in science education and present several classifications of models that have been proposed by various studies of modeling in science. I then provide an overview of a modeling cycle in science, followed by views about students' abilities for thinking in science. I describe challenges of traditional studies that claim that students lack abilities, with findings from recent studies that view students having abilities that need to be activated based on the immediate learning context.

I then turn to the idea of using CPEs as a modeling media in science and discuss their advantages over traditional modeling media and prepackaged computer simulations.

This is followed by a discussion about possible difficulties for students' views of modeling media. Lastly I provide a discussion about differences among CPEs, which was the basis for choosing the software used in this study.

1.4.1. Models and modeling in science education

Models are systematic representations of a system, or some simplified aspect of a system (Glynn and Duit, 1995), that include rules and relations between concepts and objects. They are used to describe, represent and explain the mechanisms underlying natural phenomena. Good models extend across individual systems and are complete descriptions of our understanding of fundamental mechanisms in nature. Although such models can be expressed in a number of communication media (verbal, graphical, mathematical), they are essentially conceptual in nature (Hestenes, 1997; Driver and Oldham, 1986). Conceptual models are epistemological constructs of the natural sciences aiming to provide operational descriptions of natural systems, i.e., they are interpretive representations with predictive capability.

In contrast, mental models are epistemological constructs of the psychological sciences. They do not seek to describe the underlying mechanism of natural phenomena (what, why and how it happens) but rather they seek to describe the structure and content of one's own knowledge about natural phenomena. In our current understanding of the human mind, concepts are coded into networks in the long-term memory. These networks are called mental models and although they often relate to specific situations (including, but not restricted to physical phenomena), they tend to be transient in nature (Anderson, Howe & Tolmie, 1996; Johnson-Laird, 1990).

In the physical sciences, science educators seek to establish a permanent status to conceptual models. Even though they are constantly put to the experimental test and are usually open to falsification, the more widely accepted models are established as rigorous and reliable descriptors of fundamental physical mechanisms that can be used to make valid predictions in relation to system behavior and changes (Duschl, 1990). Nevertheless, a conceptual model of a physical phenomenon, for example, is thought to represent the physical phenomenon until new evidence indicates a mismatch between the model and the physical situation, or when the model becomes non productive for supporting understanding of the phenomenon. At that point, the learner's epistemological stances should permit revision of the model to account for the new evidence.

There has been a longstanding interest in investigating the role of conceptual models and the process of constructing them in the context of science learning (Bell, Davis and Linn, 1995; Gilbert and Boulter, 1995; Kindfield, 1995; White and Frederiksen, 1998). Active construction of self-formulated models is thought to help children come to an understanding of the nature of scientific inquiry and, in particular, to appreciate the role of modeling as an ever-evolving activity of core importance to the scientific enterprise (diSessa, Abelson, & Ploger, 1991; Redish & Wilson, 1993; Sherin, 1996; Sherin, diSessa, & Hammer, 1993; White & Frederiksen, 1998; Wilensky & Resnick, 1999).

More recently, computer-based modeling media have been recognized as potentially powerful tools for implementing a modeling approach to learning in science. Several programs (for example Model-based Analysis and Reasoning in Science Curriculum project (MARS)), tried to develop curriculum aiming to develop physical concepts among students and, simultaneously, develop the abilities necessary for constructing models of physical phenomena (Raghavan & Glaser, 1995).

From these efforts, an interesting situation has emerged. Models can play a dual role in science learning: they can be both tools for learning and learning outcomes. Science proceeds through the construction and refinement of models, and learning science should include developing understanding about natural phenomena by constructing models (Constantinou, 1996; Golin, 1997), as well as learning the process of developing and refining those models (National Research Council, 1990; White & Frederiksen, 1998). Models and modeling processes can provide an approach, a scaffolding, that can guide the actual process of science learning. In this context, it is thought that the development and refinement of models can achieve better quality outcomes (in terms of fundamental understanding of concepts, operational understanding of the nature of science and the ability to employ procedural and reasoning skills) than are currently possible in many educational systems (Harrison & Treagust, 1998; Bell, 1995; Grosslight, Unger, Jay & Smith, 1991). At the same time, since models are thought to be the way we construct understanding about the physical world, they are also an anticipated result of teaching about science. As constructs of the physical sciences, they serve as free-standing learning objectives in themselves. In other words, they can be sought after as worthy learning outcomes, the end result of science education (Golin, 1997; Wells, Hestenes & Swackhamer, 1995).

Models can be used to capture students' conceptualization of natural phenomena (Penner, 2001), especially when they provide open ended, unbiased ways to talk and think about natural phenomena. They also can be "tools to think with and to reflect upon"

(Penner, 2001, p. 2), because they are representations of natural phenomena, including representations of physical and conceptual values that are not usually represented in “concrete” forms (Penner, 2001) that cannot be otherwise observed in the natural world. Thus, when a computer program procedure, for example, becomes a “thing” that is named, it can be readily manipulated and recognized by students (Papert, 1980).

In addition to being used to capture students’ ideas about natural phenomena, models can be also used as tools for constructing understanding about natural phenomena. They can help students to design (that is construct) their own representations of natural phenomena and not just to show natural phenomena. Constructing representations in the form of models in science has the advantage of putting students in a mode of work which requires them to break their understanding about what happens in a natural system into small pieces that will subsequently use to build a model of the natural system. Research (Penner, 2001) indicates that the level of observation affects one’s perception of the phenomenon: it is one thing to focus on individual objects their properties and their behavior, a different thing if you focus on two or three objects (which can possibly lead to a discussion about relationships between the objects) and yet another if you thinking about the whole system/phenomenon. Simulations, for example, provide both a view of the whole in the simulation, in addition to the parts of the model/program (Penner, 2001) and a simulation can just be described as the result of the behaviors of and interactions among several objects.

Classifications of student-constructed models in science

Models that scientists or learners construct can be classified in a variety of ways. Below I discuss three different classifications of models that are useful for studying ways that students use CPEs and the kinds of models that they construct.

Penner (2001) suggests that one way of viewing models is to distinguish between *physical and conceptual models*. *Physical* models have some kind of “external manifestation” (Penner, 2001, p.9-10) i.e., a computer-generated model of a tornado. Conceptual models on the other hand “do not depend on concrete representations” (Penner, 2001, p.10). They usually consist of laws (e.g., Newton’s law) or theories (e.g., plate tectonic theory). In this sense, conceptual models are used to support physical models and physical models are built to reflect (or at least to show) conceptual models. In this sense, as I have previously discussed physical models (e.g., paper-and-pencil drawings, 3-D structures and computer programs) are student-constructed models in an effort to construct (or capture) conceptual models of natural phenomena.

In a different classification of models, Collela, Klopfer & Resnick (2000) suggest that models can be distinguished between *illustrative models, analytical models and simulations*. Illustrative models provide visualizations of scientific processes or systems, without any reference to how these processes happen or what causes them. That is, even if there is always a mechanism underlying or causing the phenomenon, illustrative models do not provide any access to that. Analytical models on the other hand are usually based on mathematical equations that describe the phenomenon represented and they are meant to enable exploration of a variety of scenarios. In this sense, analytical models tend to generate solutions that predict behaviors of the system based on a given set of conditions. In this sense, analytical models include mathematical representations of the mechanisms of the phenomena. Lastly, simulations are models that include descriptions of the underlying mechanism that users can create or explore. In this sense, the primary goal is not only to show how the phenomenon looks (as illustrative models do), or to develop a model that can account for a variety of conditions (by enabling alterations of inputs, as analytical models do), but rather to have a model that includes the mechanism that causes and can explain the phenomenon. The difference between the analytical model and the simulation is that the analytical model utilizes abstract mathematical language, which in many cases is simply too complicated for learners to follow. Analytical models usually include some representation of the mechanism that underlies the phenomenon, and even if this might be accessible to the users, mathematical language could simply make it difficult to read. In a way, analytical models may be seen as similar to prepackaged computer simulations which are not meant to give access to the code that generates the simulation.

A third possibly useful classification of models is provided from Löhner & van Joolinger (2002), who talk about different ways that students use CPEs to develop models of natural phenomena. According to their classification modeling can be *expressive and explorative*. In expressive modeling learners externalize their thoughts about a particular domain (e.g., of science) or a phenomenon by creating a model. The focus is on communicating ideas about the phenomenon, rather than constructing detailed models of the phenomenon or understanding in detail what is the mechanism that causes the phenomenon. For instance, expressive modeling can be used when talking about molecular systems in biology, in which general rules apply and can roughly describe the process that take place, without getting into the specifics. In explorative modeling learners try to develop a specific model of a given domain (e.g., of a particular phenomenon), seeking to find the rules that govern the phenomenon. In explorative modeling learners seek to develop representations of the phenomenon under study, and they are meant to "demonstrate" a detail understanding of the phenomenon.

1.4.2. The model-based learning cycle

One approach to engage students in the practice of constructing and refining models is the "*model-based learning cycle*" (Constantinou, 1996; Louca & Constantinou 1999; Louca & Constantinou, 2002). This has two major stages: the *model formulation stage*, and the *model deployment stage*.

The model formulation stage

During the model formulation stage, the learner develops a model (Constantinou, 1996) by studying the natural phenomenon and collecting evidence from the real world (Bell, 1995). Prior experiences are crucial for this stage, as well as the ability of the learner to identify and use those experiences that are relevant with the particular phenomenon under study. Once students have constructed a model, they are asked to examine the relation between the model and the real world. The comparison of the model with the real life situation is the driving force for its iterative refinement (Bell, 1995) and it is an important part of the cycle (Louca & Constantinou, 2002).

The model formulation stage of the model-based learning cycle is equivalent to the first couple of "model phases" as described by a number of researchers (Penner, 2001; Penner, Lehrer, & Schauble, 1998; Schecker, 1993). They indicate that a first crucial step is the identification of the need to describe, predict and explain a natural phenomenon, which will then guide the learners to investigate the phenomenon and develop a model to represent it. Then, as Constantinou (1996) indicates, the learners use their current understanding (such as observations from everyday or lab-based experiences) to simplify the natural world into objects and their interactions to be represented in a model. This leads to the construction of a model and it is an important step because it includes identification of the model's components, and relation between the components (Schecker, 1993)

The model deployment stage

The second stage of the model-based learning cycle, model deployment, begins after students have constructed what they consider a satisfactory model. In this stage, students attempt to apply their model to new situations. This process requires them to use the model to interpret some phenomena, and also to make predictions about others. If the model is not successful in deployment, students return to the development stage.

The deployment process also requires that students use the model not only in interpreting phenomena, but also in making predictions about other ones. The modeling process does not end here. So far, students have been developing simple models – interpreting parts of the physical phenomenon. This implies a step-by-step development of the model from a simpler stage to a more advanced level (Golin, 1997). "…complex theories in science are developed through a process of successive elaboration and refinement in which scientific models are created and modified to account for new phenomena that are uncovered in exploring a domain" (White and Frederiksen, 1998, p.7). The idea is to keep the model-based learning cycle going with testing, revising and re-evaluating the constructed models (Bell et al, 1995).

Penner (2001), Penner, Lehrer, & Schauble (1998) and Schecker (1993) indicate that after the development of the first model, the learner needs to evaluate it by comparing its behaviors with the phenomenon under investigation. The purpose of the evaluation is not to identify whether the model is right or wrong but whether it accurately represents the behavior of the phenomenon. Evaluation of the model leads to subsequent modification of the model.

Data from a previous study (Louca & Constantinou, 2002) indicate that students come to understand the purpose of the model-based cycle as a process of development and continuous refinement of their understanding. Findings from that study suggest that both stages of the model-based learning cycle are important for learning in science through the modeling process as well as learning about the process itself. However, in that study, the researchers did not focus on the process of using the modeling procedures; our focus was primarily on the importance of such learning for children’s understanding in physical science. It is equally important, however, to study the characteristics of children’s work while using modeling as the learning process.

1.4.3. Views about student’s use of prior knowledge and thinking strategies

Modeling as a learning process in science requires a variety of inquiry strategies among learners. Developing a representation of the mechanism that underlies a physical phenomenon is central to the process of learning with models. This mechanism should provide an explanation of what is happening in the situation and more importantly how it happens and what are the elements of the phenomenon’s underlying mechanism. To do that - as identified in the model-based learning cycle - students are expected to use relevant prior experiences (from a repertoire of experiences that they have both from science classes and the real world) specifically to reconcile different points of view as well as different models about physical phenomena. Learners should also be able to use prior experiences to support or contrast models developed about natural phenomena as part of the process of model deployment and model evaluation. Thus, theoretical argumentation is another important thinking strategy for modeling practices. Lastly, the ability to develop or select a theory that would explain a natural phenomenon is another inquiry strategy that seems to be important for modeling practices.

There is currently a debate about whether students are developing such thinking strategies as part of their cognitive development (Kuhn, 1989), or whether Piagetian accounts of these abilities underestimate students’ thinking strategies (Metz 1995). The first view, which follows a developmental perspective, suggests that students are expected to develop particular abilities as part of their cognitive development; thus there are ages that students should not be expected to have and use particular thinking strategies. In the second view, students are thought to have a repertoire of thinking strategies that are seem to be dependant on several factors (including context) and as such would not be used in every learning situation (Louca, Hammer & Bell, 2002; Samarapungavan, 1992).

One perspective for studying student thinking has been provided by Deanna Kuhn’s work (1989, 1993, 2001). Kuhn (1989) has called attention to inquiry as an essential objective for science education. She particularly called attention to argumentation as an inquiry strategy in science itself and as an objective for science learning.

In that perspective, Kuhn (1989) argues that there is ample evidence that students lack thinking abilities about argumentation in science. For instance, students below 6th grade seem to be unable to differentiate between theory and evidence in science. They have been observed to fail to coordinate different perspectives about scientific evidence of particular phenomena, and thus fail to differentiate between different and sometimes contradictory theories. In this view, 9th grade students have been reported to fail to make distinctions of covariation or non-covariation of evidence with a theory (Kuhn, 1989).

The explanation that Kuhn (1989) provides about the differences of students' and adults' thinking abilities is based on the notion that these abilities are developmental in nature and thus young students (of the elementary school for instance) have not yet developed those abilities. Kuhn (1989) suggests that it is rather a matter of waiting for (or in another view perhaps accelerating) the development of these abilities among students before they would be able to use them.

It is possible that the above view of developmental limitations in student thinking abilities may underestimate student's inquiry strategies in science (Metz, 1995; Louca, Hammer & Bell, 2002; Samarapungavan, 1992). "Recent studies have shown that when tested with tasks that are simple and meaningful, children demonstrate an impressive array of skills related to scientific reasoning" (Samarapungavan, 1992, p. 3). These skills would include abilities to think causally, analogically and inductively (Brown, 1990; Vosniadou, 1989; Gelman & Markman, 1986). Similarly, data from other studies (Louca, Hammer & Bell, 2002; Karmiloff-Smith & Inhelder, 1974) indicate that given enough time, students can use a variety of thinking strategies that developmental views would fail to attribute to students of that age. In this sense, Samarapungavan (1992) argues, these research findings "demonstrate the importance of tasks that tap into children's knowledge of the world" in an attempt to activate thinking resources that might be readily available but inactivated. Further, findings from her study (Samarapungavan, 1992) suggest that when metaconceptual criteria for evaluating explanations are made "salient" to students, they can use them successfully. Whether this use can be spontaneous, still remains an open question – and an important one.

In this study, I share the assumption that students have a repertoire of thinking abilities that they can use when learning in science, depending on the particular context of the learning situation. Since modeling practices require abilities like the ones aforementioned, it is reasonable to argue that students may have sophisticated inquiry strategies for modeling that science educators need to tap into when using modeling practices as means for learning. Starting from that assumption, I seek to describe what thinking strategies students might use during modeling in science using CPEs and what tools provided or working contexts supported by CPEs can tap into abilities, for example for theoretical inquiry or modeling.

1.4.4. Computer programming tools as means for developing modeling skills

In modeling practices as means for learning in science, the process of model development and deployment may be compared to the process of writing and implementing a computer program. Most powerfully, it can be carried out through a computer program, when the program itself becomes the scientific model. In this way, the program language becomes the design medium for the scientific mode and the program (outcome) becomes a way of clearly articulating one's understanding about scientific phenomena. This has been the approach of a number of educators interested in computer-based modeling and science education (diSessa, Abelson, Ploger, 1991; Louca & Constantinou, 2002; Redish & Wilson, 1993; Sherin, 1996; Sherin, diSessa, & Hammer, 1993; White & Fredriksen, 1998; Wilensky & Resnick, 1999).

Programs in CPEs produce *a computer microworld* which is a structured environment that learners can use to explore and manipulate a rule-generated universe, subject to particular assumptions and constraints that serve as analogical representations of aspects of the natural world (Pea, 1984). Computer microworlds are idealized environments composed of objects, relationships among objects and operations that transform the objects and their relationships (Thompson, 1985). diSessa (1982, 1988) indicates that microworlds can provide learners with opportunities to manipulate realities in ways that learners cannot do in reality.

One of the difficulties that students confront is *understanding the relationship between a scientific model and "reality."* Inevitably, scientific models neglect aspects of the physical world, which troubles students who expect that scientific models are "true." Students confront difficulties in accepting that the media they use for modeling representation require them to develop models that include part of the physical reality. However, in the context of writing a computer program, including a program to simulate the physical world, students tend much more easily to accept the notion that the program cannot reproduce everything, and thus the programmer must select which aspects of the world to represent and which to ignore (Sherin, diSessa & Hammer, 1993). The computer programmer usually starts with an environment that has no rules and follows no physical laws; the programmer has to identify and create them. The microworld will then function according to the designed rules that represent physical laws. In representing those aspects, it is necessary to make idealized assumptions and approximations, and much of the challenge lies in judging what assumptions are possible and useful. In this way, the task of writing a computer program consists of creating an idealized "possible world," as Medawar (1987) described the process of constructing scientific models.

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 also need to learn the practice of quantitative precision: For an idea to be useful in physical science, it should be made sufficiently precise in order to maintain consistent meaning across different contexts. That would also enable the idea to constrain interpretation sufficiently, so that different people who apply the idea in different contexts will arrive at the same result (Hammer & Elby, 2003). In this sense, it is possible to argue that the process of programming can enhance the development of refined meanings of physical concepts and also can promote the use of precise language to provide explanations about natural phenomena. The importance of this precision also lies on the fact that it is related to students' understanding about the mechanism that underlies the physical situation under study. For instance, students may often have difficulty articulating orally the mechanism that underlies a physical situation. This is not to be interpreted as a lack of this ability. Rather, it is possible that these thinking resources are context dependant. Through this study I investigated whether and how modeling through programming might serve as a learning environment where children can use reasoning resources, and thus examined what are the characteristics of such learning environments.

Having to design a microworld to illustrate for instance the concept of speed of a particular object, students have to *deconstruct their understanding* of the particular physical mechanism into small programmable pieces of knowledge. Pieces may include the mechanism that makes changes to the speed of the object (i.e., acceleration), speed's relation with the motion of the object (speed can he related with the amount of <forward> in a program) and possible relations between one object's speed to another's (e.g., in a relative motion situation). The necessity of this deconstruction is created by the need to transform an idea in science into specific, technically precise programming code. To do that, they need to use precise "program language". In this sense, program language can be a useful tool for understanding and thinking about science. Programming like mathematics (which is a principle tool in science as it provides a language for formal, technical precision and consistency) can be an alternative language for using in developing understanding in science (Sherin, 1996).

Programming has at least two important advantages over any other written form of language such as mathematics. First, unlike mathematics, *a program can be run* on a computer and its results observed, allowing an iterative process of testing and debugging that may be more tangible and accessible for young learners than the iterative process of developing and deploying a scientific model expressed in 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, thinking through what the computer is being told to do. Towards this end, students can use the program language as a clear and precise way of developing understanding in physical science and also communicating their ideas with others.

Using programming tools for constructing models in physical science has also several other advantages. CPEs encourage exploration and *"exploratory" learning* (Smith & Cypher, 1999). Providing an open-ended programming environment, in which students can easily investigate different possibilities and different combination of things, programming tools promote creativity (Smith & Cypher, 1999). In fact computer capabilities can offer students various possible ways to provide answers to their questions; that is, to construct several models (that differ in type and complexity) that will provide explanations for the natural phenomena.

The use of CPEs may also have another advantage over other modeling media in helping to tap productive resources among learners. Since CPEs can be viewed as learning environments, it is possible that as such they can tap particular thinking resources among students. One example is *debugging*, which is a characteristic of programming that is important for learning in science in the following sense. Debugging in the programming process mainly consists of the identification of the reason(s) that programs do not function the way the learners think they have developed them. Thus, in order to debug, programmers/learners have to coordinate two different perspectives: their own (that would be what they intended to accomplish by the particular code they wrote) and the ones actually represented by the programs they wrote. The flaws can be either due to typos or to mismatches between the students' ideas and the programming codes that the students used to represent their ideas. For debugging, students will have to identify the typo or the mismatch. In this sense, programming as a learning environment supports by default the process of debugging and thus programming as a learning tool can tap into useful and productive thinking resources that are possibly otherwise inactivated.

Differences between CPEs and prepackaged computer simulations

Programming tools that I investigated in this study differ from prepackaged computer simulations of natural phenomena in the following sense. In prepackaged computer simulations, students are provided with an already constructed simulation (without any access to its code) of which they can explore and possibly alter some of its parameters. However learners do not deal with neither the development of the simulation nor with what concepts are embedded and how are represented in the simulation.

Penner (2001) indicates that pre-structured simulations are biased ways of providing students with learning opportunities to investigate natural phenomena, because they already have a particular structure, are meant to be used in particular ways and students do not have access to the code that creates the simulation (as opposed to open-ended CPEs where students can develop their own representations of natural phenomena). There is an assumption that underlies the above idea of providing students with pre-constructed "experimental apparatus" in science education (including hands-on experiments and simulations) (Penner, 2001), that science thinking, as it is partly described by NSES (1990) and theories of cognitive development, is a set of "semi-independent cognitive skills" (Penner, 2001, p. 9) that students need to develop, such as hypothesis generation, experimental design, hypothesis revision etc. This view contradicts recent views about students already having developed abilities for theoretical reasoning, or argumentation in science, that only require activation.

The programming tools of the type that I used in this study, in contrast, require that the *learners participate in developing and debugging the code to generate a simulation.* That is, 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" (Collela, Klopfer & Resnick, 2000, p.1)

1.4.5. Difficulties derived from the modeling process

Modeling media may contain at least one common limitation that can pose possible difficulties to the learners. Students may be more inclined to use the particular media in ways with which they are familiar, which are different from using those media for modeling purposes. Many of the modeling tools that research in science has used for teaching about and with models are tools that children are usually familiar with in other settings. These would be paper-and pencil tools (diSessa, Hammer, Sherin & Kolapakowski, 1991; Louca & Constantinou, 2002), 3-D structures (Penner, Giles, Lehrer and Schauble, 1997), which may include physical experimental settings (Louca & Constantinou, 2002), and CPEs. The challenge lies in getting students to use these media as modeling tools and not as simple visualization tools. Previous research in science/model-based learning has not documented this as a difficulty. More importantly this issue has not been identified as a possible limitation of computer-based modeling, despite the fact that such difficulties have been identified in a number of modeling studies with young learners, with a variety of modeling media (for example see diSessa, Hammer, Sherin & Kolapakowski, 1991; Penner, Giles, Lehrer & Schauble, 1997).

In their research, Penner et al (1997) asked students to design 3-D models that would represent the function of a human elbow. In the beginning of the study, students were focused on how to design their 3-D models to *look* like the real human elbow rather than *function* like the human elbow. In this case, modeling media were perceived and used as ways to represent reality and not to represent the function underlying that reality. Similarly, in their study, diSessa et al (1991) asked students to draw a representation of an incident in which a person while driving in a desert passes a cactus. She then stops, turns back, drinks some water from the cactus and resumes her trip. Students' initial drawings were showing only a desert and the cactus, capturing the visual context of the situation and failing to represent the actual motion of the driver in her car, which was the purpose of the activity. In a way, it can be argued that students perceived and used modeling media in this case (paper and pencil) in a familiar way, to represent still pictures rather than represent processes.

In both the above cases, students were using modeling media in ways they were accustomed, rather than using media to develop a model that would represent the physical situation. Activities in these cases were perceived to have the goal of making visual representations of the reality and not representing the function(s) of objects in the phenomenon under study. Students, I am arguing, may not be accustomed to using particular modeling media for modeling purposes, even if they might be familiar with modeling practices. Thus, they are inclined to use the media in ways with which they are familiar with and they have used before.

However, it would be naïve to argue that students lack the appropriate resources for using modeling media (such as 3-D structures, paper and pencil media or CPEs) for designing, constructing and evaluating models, simply because this would underestimate students' abilities for reasoning and thinking in science. In fact, the aforementioned studies have shown that tapping into students' particular resources that support modeling practices is an easy agenda. Recent findings about student abilities (see section 1.4.3) suggest that students may have a repertoire of abilities to think e.g. in science; their abilities however are context dependent and need to be activated to be used.

For example in the above modeling studies (diSessa, et al, 1991; Penner, et al, 1997), students failed to spontaneously use modeling media for constructing models of physical phenomena. For example, For instance in their research, Penner et al (1997) asked from students to design 3-dimensional models of a human elbow. In the beginning, students were focused on how their model *looked like* (and thus how to make it look more like the real human elbow) than on how to design a model that would *function like* the human elbow. Similarly, in their study, diSessa, et al (1991) asked their students to draw a picture representing an incident in which a driver while driving in a dessert passes a cactus. She then stops, returns back, drinks some water from the cactus and resumes her trip. Students' initial drawings were showing the dessert and the cactus, and were not a representation of the actual motion of the driver in her car that was the purpose of the activity. However, when this issue was addressed, students could use media for modeling purposes in both above cases. This is a limitation, I argue, of the modeling media, especially for the media that are not specifically designed to be modeling tools. However, for CPEs that are designed to be used as modeling tools, it is reasonable to expect that they should be designed in ways that can tap into abilities for scientific inquiry that are useful for modeling natural phenomena. This would provide CPEs with an important advantage over other modeling media: the advantage of being easily perceived as modeling tools.

However, whether we should aim to design CPEs to easily tap into these kinds of modeling resources among learners is still an open question. In the previously discussed modeling studies, (Penner et al, 1997; diSessa, et al, 1991; Louca & Constantinou, 2002), researchers used modeling practices as simple as questions to guide students towards the modeling process, that would include designing, evaluating and then improving a model. These modeling practices can help students to conceptualize modeling media as the tools for the modeling process. An approach that includes instruction about the modeling practices themselves provides an additional opportunity of learning about the process of modeling itself, in addition to providing children with the tools to tap into their modeling resources by themselves.

There are two important issues that stand out from this discussion. First, the need for modeling media that would easily promote modeling thinking strategies to be activated by the students is apparent. Students, I have argued, are reasonably expected to be familiar with some of the processes of modeling, especially if the processes are related to a context with which students have extended experiences. Secondly, it is equally important that students not only use modeling practices but also that they are aware that they use them. Investigating whether programming environments designed for children can support collaborative modeling practices is still an open question that I partly investigated in this study.

1.4.6. Different types of program languages

Currently there are a number of software programming environments available for children. CPEs that have been developed for young children share several common characteristics. They also include a number of features that are different among different environments. Programming environments of the type that I used in this study differ in the program strategies the user is required to use, the program language that they use, the representation of the objects, and the representation of the physical values. Below I discuss some of the better-known applications focusing on their basic characteristics.

As mentioned before, computer applications developed for making programming easier for children share similar characteristics designed to promote modeling and thinking strategies for students. Research (Smith & Cypher, 1999; Sherin, diSessa & Hammer, 1993; Louca & Constantinou, 2002) has provided examples of students using CPEs to construct models in science: children explore the programming tools available to them in order to find solutions for their program goals. In this light, programming for young learners can be an exploratory experience, which promotes creativity.

However, different types of programming require different types of program strategies. Programming can be done by demonstration (such as in Stagecast Creator) (Smith and Cypher, 1999) using “click-and-drag” techniques. The user creates a script that the computer can monitor and model so that it can be performed later. Figure 1.1 provides an example of such program techniques. Rules in this software are visual “if-then rules” rules (Smith & Cypher, 1999). That is, the user can design rules in the form of “if-then rules”. For a given situation, an action can be determined.

In a more advanced programming environment, children can create more general scripts by choosing either language or demonstration as a medium for programming. Students can also create variables indicating the different behavior they expect to observe in different situations. An example of this kind of programming can be found in Toontalk (Kahn, 1999). This programming environment is “a video game for making video games”. In this case, the user designs what is considered to be virtual worlds in which children can reassemble situations from real worlds: learning here is by playing.

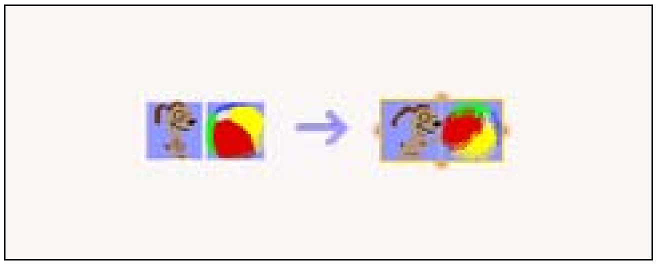


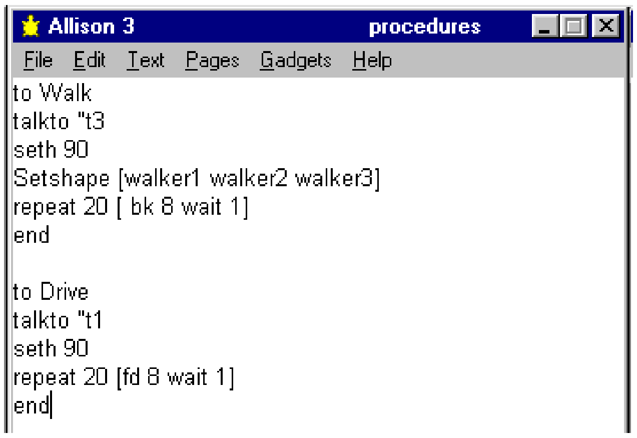
Figure 1.1. Program example from Stagecast Creator: If-then rule

Program language also varies among different programming environments. Microworlds Logo, for example, (a revised version of Logo (Papert, 1993)) uses formal program language as the medium of expressing relationships about (virtual) objects. As indicated before, a formal program language provides the means for designing very accurate mathematical models of the physical world. Microworlds Logo also includes more graphical representations, the capability of multiple characters simultaneously running under the program; interactions between characters and among characters and the mouse are also possible. Figure 1.2 provides an example of programming in Microworlds Logo. Further, Microworlds Logo includes the capability of easily importing sound, movies, pictures as well as different characters and the feature of easily creating animations as part of the programming process. However, the visual capabilities of the Microworlds Logo programming environment are limited to the outcome and not the programming process itself. For instance, programming a simple code that would enable a character to walk requires entirely formal program language: graphical representation is not a part of the programming process. On the other hand, the same code in Stagecast Creator is entirely based on graphically represented language. Other programming environments enable a combination of formal program and graphically represented language.

The process of programming in Microworlds Logo is *procedural*: the user needs to write instructions, one after the other for the turtle (character) to follow when the program is executed. On the other hand, programming in Stagecast Creator is *object oriented*: the user has to assign each character with rules that define its behaviors, which the program would execute when the conditions of each rule are met.

The representation of objects also varies. Stagecast Creator uses analogical representation: an object is represented in the same way in all different levels of the programming environment (the program level, the outcome level etc). This way, the graphical environment that is used allows direct manipulation of the represented objects and easy assignment of rules to each object (Smith & Cypher, 1999). On the other hand, in Microworlds Logo objects are represented by an image in the output window and by a "name" in the program window. For this reason, even though most programming applications are object oriented (that is every object has its own identity and thus can be manipulated by the programmer independently), the way of creating, running and debugging program varies both in difficulty and complexity.

The representation of physical values also varies. In Stagecast Creator the programmer can create boxes named after variables (such as “energy level” of the characters) and design rules to add to or subtract from these variables depending on the behavior of the characters. In addition, in the same software package, the programmer does not have to set the rate of any action (such as velocity of motion) because rate is set as an external characteristic of the programming process, pre-set to depend on the speed of the microprocessor cycle in microcomputers. Of course there are several ways that the programmer can write a code for identifying velocity, but this is not the easy default way that the software was designed to work. On the other hand, in Microworlds Logo for example, the changes in a particular variable are a result of a mathematically precise code that is expressed in the program languages. Further, in Boxer (diSessa, Abelson, & Ploger, 1991) the same code is represented by a combination of the two different ways.



```
to Walk
talkto "t3
seth 90
Setshape [walker1 walker2 walker3]
repeat 20 [ bk 8 wait 1]
end

to Drive
talkto "t1
seth 90
repeat 20 [fd 8 wait 1]
end
```

Figure 1.2. Program example from Microworlds Logo: Textual program language is used

In Stagecast Creator, for instance, if the programmer places a character releasing a ball in the output window of the software, the program will not “know” what to do with the released ball. As a first step, a rule can be created that would enable the ball to fall towards the ground. As discussed before, programming here is done by indicating to the software that in every “cycle” the ball should be moved a “distance unit” towards the ground. Further, the time for each action (i.e., time that a particular action is carried out) is set by default to be the rate of the cycle that the Stagecast Creator utilizes and thus different speeds (rates of motion) should be explicitly defined as different distances traveled in the same time. This however, can have implications for further steps in programming. One way to make a ball falling to speed up, is to simply create a number of rules that will be executed sequentially; each subsequent rule will result a larger distance traveled in the same time. Instead of this, however, in a second step, the programmer can develop a variable “velocity of the ball” and create a mathematical relationship for the distance falling in each rule with the total time of the motion, such as increasing distance traveled in each cycle by 5 units every cycle (or “second”) of the program. In sum, it is possible that the design of the software does not prompt students to specifically identify the rate of the speed, (but rather making it easier to create a number of independent rules that can simply recreate the phenomenon) leading to inaccurate representation simply because students have just not considered to use.

In Microworlds Logo, however, in creating a rule, the programmer has to provide information about the velocity of the action. Similarly to Stagecast Creator, the program runs the rule in cycles (e.g., the character will move the defined distance every cycle), but in Microworlds Logo, only the rate of the cycle is defined – not the rate of the actions themselves. Therefore, in this case students need to provide the speed of the motion, or the mathematical function that would describe the change in the velocity of the falling ball. I argue that because the programming environment does not provide the speed of the actions, students are prompted to provide that information.

Given the differences in the ways that CPEs have been developed to be used by young learners and the different characteristics that they have, it is necessary to define which characteristics are useful for young learners in science. It is necessary to define those characteristics (that such software titles may share or not) that address particular programming needs and learning habits of students in science and how different characteristics of different software enhance scientific visualization. It is equally important to learn how the limitations of the available software programming tools affect learning in science. Finally it is necessary to understand the ways that those programming environments provide metacognitive knowledge to the students using them.

2. METHODOLOGY

2.1. Methodological approach

This study has followed the qualitative research tradition for classroom based studies for two major reasons. First, the study is based on the idea that research in the form of clinical interviews and pre and post-test designs fails to capture the dynamics of the regular (science) classroom. It also fails to focus on students' activities and conversations during the process of learning, because the focus is on what students can re-produce during interviews and testing. On the other hand, qualitative research methodologies focus on the collection and analysis of large amounts of data that can derive from the actual learning process and can be analyzed using a variety of methods. In this way, the focus is on the activities during the learning process.

Second, the purpose of this study is to develop detailed descriptions of students' work with computer-based programming environments (CPEs), and the ways that students may or may not use CPEs to support their learning needs. In addition, the focus of the study is on students working patterns with CPEs in school settings. For all these reasons, this research follows the qualitative research tradition.

2.1.1. Characteristics of qualitative research

Qualitative research methods share common characteristics about the nature of the data sources and the way researchers develop their understanding of them.

<u>Qualitative research is naturalistic</u> (Bogdan & Bilken, 1998). It is focused on the natural settings where the phenomenon or the situation under study takes place, rather

than recreating those settings in the controlled environment of a laboratory (Bogdan & Bilken, 1998). Being naturalistic, qualitative research addresses two major concerns. First, the focus of the study is the actual setting in which the phenomenon under study takes place. Context is believed to be an important factor of the situation and without that, parts of the phenomenon's complexity may not be captured or recreated. Second, the behaviors of study participants are believed to be related to its natural context where the phenomena of interest usually take place. Researcher-controlled situations may fail to accurately and successfully recreate the natural behavior that is intended to be studied and thus natural situations are chosen to be studied by qualitative researchers (Bogdan & Bilken, 1998).

<u>Qualitative research is descriptive</u> (Bogdan & Bilken, 1998). Both data collected and reports developed are rich descriptions of the situation or phenomenon under study (Bogdan & Bilken, 1998). Being naturalistic, qualitative research seeks to develop detailed holistic descriptions of the field and the phenomenon/situation studied. The focus being on the whole situation under study, qualitative researchers try to provide rich descriptions of the situation. For this reason, considerable time is spent in describing the context of particular setting of the situation under study (Creswell, 1988). It is important to describe the larger picture in detail, which would provide important information to situate the situation/phenomenon within its context.

<u>"Qualitative researchers are concerned with the process rather than simply with outcomes or products"</u> (Bogdan & Bilken, 1998, p. 6). The purpose of qualitative research is to describe and document the process that takes places within a

phenomenon/situation in its natural environment. Outcomes of the process can be parts of the data, but are not the major purpose of qualitative studies.

<u>Qualitative research is inductive</u> (Bogdan & Bilken, 1998). Data of qualitative research are analyzed in inductive ways; researchers seek to develop claims about the phenomenon/situation under study that can be justified by their data (Bogdan & Bilken, 1998). The purpose of qualitative research is to provide a possible theoretical perspective that would be grounded in the collected data. As Bogdan & Bilken (1998) indicate, qualitative researchers do not put together puzzles whose pictures they already know rather they are constructing pictures that take shape as they collect and examine pieces of data.

<u>Qualitative research is concerned with meaning</u> (Bogdan & Bilken, 1998). Qualitative researchers “are interested in how different people make sense of their lives” (Bogdan & Bilken, 1998, p.7) or more generally how they experience the phenomenon/situation under study. In this sense, qualitative research seeks to capture the views of people in the study about the phenomenon and how they live the situation that is being studied. Without limiting qualitative research in investigating the sense that people develop or have about different situations, and while being interested in the process and not the outcome, qualitative research also seeks to develop descriptions of the ways people make sense of the phenomenon/situation under study.

2.1.2. Case study tradition

This qualitative study follows the case study tradition. A case study examines a particular phenomenon, situation or event, concentrating on as many as possible variables that are involved in the particular case (Creswell, 1988; Merriam, 1988). Being holistic in nature (Merriam, 1988), a case study seeks to define, intensively describe and interpret the case under study, using as much detail as possible and available, so that the picture provided would be full and holistic.

Case study research requires *identification of the "case"* that is under study. As such, the qualitative case study research method focuses on a "bounded system" (Creswell, 1988; Merriam, 1988). Boundaries are of time and place. That is, the case should be defined within particular time frames and be in a particular place (Creswell, 1988). The focus of the study is on the phenomenon on its whole, and studying it does not consist of a linear model of inquiry. This is to stress that there are complex relationships within phenomena, that taking them apart may result in losing some of their important aspects.

Qualitative case study research addresses "*how*" and "*why*" research questions (Creswell, 1988; Merriam, 1988; Yin, 1994). It tries to provide a detailed description and analysis of the observed case. Case study is also *naturalistic* in the sense that it studies cases in their physical context, in which the researcher is also interested (Merriam, 1988). For this reason, the case study method requires the study of the "environment" that constitutes the "bounded system" and thus, the researcher has limited or in some cases no control over the case of study (Merriam, 1988; Yin, 1994). Case study method also requires the study of a contemporary phenomenon or situation within its real-life context, so that the researcher can be the primary observer (Yin, 1994).

Overall, this study is an interpretive case investigating “how do fifth grade students use CPEs?” The study includes investigation of conceptual categories (i.e., student thinking strategies) and theoretical assumptions (i.e., students have a repertoire of resources that they can use when talking and thinking about physical phenomena). For this reason I have provided a theoretical framework that situates this study in the area of research in science education and model-based learning. Further, I describe in detail how fifth grade students use CPEs in the context of learning about physical science, following and analyzing almost two months of weekly afternoon meetings. I then use these descriptions to analyze students’ work characteristics while using CPEs as learning tools.

One of the important issues that influenced my decision to use qualitative case study methodology for this research is that there is not sufficient foundation yet for more quantitative research in the area of studying students’ use of CPEs in science. There has been a long history of interest in the use of models in science (diSessa, Abelson, & Ploger, 1991; Redish & Wilson, 1993; Sherin, 1996; Sherin, diSessa, & Hammer, 1993; White & Frederiksen, 1998; Wilensky & Resnick, 1999). Most of these studies, however, investigated the use of CPEs with older students. Further, traditional research with programming media (usually investigating the use of Logo) studied the development and transferability (to other domains) of skills such as problem solving skills, planning skills (e.g., Orhun, 1993; De Corte et al, 1993; Enkenberg, 1989; Verschsffel et al, 1989). Some studies have also focused on cognitive issues that derive from the use of Logo, but studied the difficulties that students encounter with the Logo program language (e.g., Fay & Mayer, 1987). In these studies, however, researchers were focused on students’ “misconceptions” about Logo program primitives and used pre/post test designs to investigate how students, for example, predicted the output of particular programs in Logo.

In this study, I seek to contribute to our understanding about how students work with CPEs in the context of developing models/representations of natural phenomena. Specifically, I wanted to investigate the ways that students use tools available in CPEs, and what kind of thinking might those tools support among young learners. Due to the limited research foundation around this area of research, studying cases in a qualitative way enabled me to discover aspects of the phenomenon I would not have through more narrow, instrument-based data collection. Further, one of my goals was to provide detailed descriptions of students' work with CPEs, and a qualitative research approach provided me with the tools to collect, analyze and present such descriptions of students' work with CPEs in science.

2.2. Study setting

2.2.1. Site of investigation

This study took place at a suburban elementary School in Montgomery County, Maryland. In collaboration with the school principal, and after getting approval from the University IRB and Montgomery County, I set up an afternoon science/computer club for fifth graders, where students learnt and used Microworld Pro and Stagecast Creator for developing models of physical and/or biological phenomena.

In February 2002, I contacted the school's principal, asking for his possible interest in my study and for setting up an afternoon science/computer club for four months at his school. The school principal indicated his interest in the study and in March

2002 we finalized the setup of the study. In June of 2002 the school distributed information for the club for the following year, asking parents who would be interested in having their children participate to inform the school administration. In September 2002, the same information was redistributed and additional students indicated their interest in participating. The school administration formed a list of the interested students, and the first 30 students were selected to participate in the study.

Because of the nature of the study and my goal to study the use of two separate CPEs, I formed two separate teams of students that met in different days (one team on Mondays and the other on Wednesdays). Each team of students was taught and used one of the two CPEs: the Monday team used Stagecast Creator (SC team) and the Wednesday team used Microworlds Logo (MW team).

2.2.2. Study participants

Students selected for participation were divided into the two teams based on their indicated preference of the day that they wanted to participate in the study. The teams were representative of the population/cultural diversity of the school, and they included five[1] African-American students (three girls and two boys), two Latino students (one girl and one boy), one Chinese student (boy) and 11 Caucasian students (two girls and nine boys), a total of 19 students, six of which were girls and 13 were boys.

Students in the school that the study took place had some experience with computers, even though none of the participants had previously used either software that were used in the study. The school has a computer lab, and a designated computer teacher

[1] Numbers reflect students that remained until the end of the study.

with a teacher assistant. Students in the school regularly visit the computer lab, where the computer teacher in coordination with the students' regular teacher prepare lessons in particular subjects (e.g., American history, social sciences, etc) that involve the use of computer applications (e.g., PowerPoint presentations, search over the Internet etc.). In this sense, students participated in the study had previously (in some cases quite extensive) used computer applications within the school and were accustom of using mouse, navigating in the world wide web, retrieving documents from and sending documents to my (teacher's) account. Further, during their work in with the computers, students were following guidelines that were established by their computer teacher for work with computers (such as, logging in with their own personal password, always log off before leaving, and take turns in using the computers).

The club met with me once a week on Mondays or Wednesdays, for one hour after the school ended (from 3:30pm – 4:30pm), from September 9 through December 16, 2002. During the first phase of the study (see description later in this chapter) students worked in 5 groups of 3 students each. Around mid October, several students from both groups stopped showing up for the afternoon club. There was no formal communication (from neither the students nor their parents) about the reasons that students stopped coming to the club, but their classmates indicated that some had conflicting schedules or switched to other clubs in the school.

For the SC team 10 students in total remained until the end of the study; 2 of them however were not participating regularly and were also not providing any justification as to why they were missing the meetings. For the second phase of the study (see description later in this chapter), for the SC team, I divided all regularly participating students in 4 groups and the 2 students that were not participating regularly in 2 of the above groups. Thus, 2 groups had 2 students and another 2 groups had 3 students. In the MW team, 9 students remained and were attending regularly until the end of the study. Therefore, for the second phase of the study, I divided them in 4 new groups: 3 groups had 2 students each, and one group had 3 students. In all the cases, the distribution reflected both cultural and gender diversity among the groups. However, the most important factor for putting students into groups was to have groups of students who could work together and communicate successfully. Thus, during the first phase of the study, I switched groups of students quite regularly (usually twice a month) in order to find out about possible combinations of groups that could have been productive. In the second phase of the study, depending on their ability to work with others, some groups had members of different gender and in other groups the members were of the same gender.

2.2.3. Study's computer-based programming environments

For this study I used two different CPEs designed for young learners. These are: *Stagecast Creator* and *Microworlds Logo*. Given the large numbers of available programming environments that have been specifically designed for young learners, I came across the challenge of choosing what software packages to use in this study. For this reason, I established several criteria for choosing the two software packages for the study. The first criterion was based on the existing literature and research about programming environments in general, and programming environments for young learners in particular. Most literature, however, does not document in detail the particular ways that students use the different programming environments (for example see

Underwood et al, 1996; Rader, Brand & Clayton, 1997; Singh, 1992; Pea, Krland & Hawkins, 1987; Leher, Lee & Jeong, 1999). Traditional studies usually study the effects of using programming for developing abilities such as for problem solving among learners using pre/post tests designs. However, there is some literature that provides descriptions of the characteristics and capabilities of software, mostly derived from the process of developing and testing prototype systems (i.e., Smith, Cypher & Telser, 2000; Cypher & Smith, 1995; Rader, Brand & Lewis, 1997).

Singh & Chignell (1992) provide a discussion about programming media and suggest that there are two major types of programming media. Their differentiation is based on the type of programming language that CPEs use. The two major types of CPEs are those that use traditional textual programming language and those that use visual programming language. The type of the programming language has implications on the programming process: textual language is a more open system, enabling users to create many kinds of routines, whereas visual language restricts users to pre-defined scaffolding for creating programs. This was my first criterion: investigate the use of two different types of programming environments.

My second criterion was based on my previous experience in using CPEs with young learners. I did not want to start with this study without having any idea of how students would use them or what the study would look like. Both in my pilot study (findings reported in Louca *et al*, 2003) and a previous study in Cyprus (Louca & Constantinou, 2002), I had used SC and MW successfully with 5th and 7th graders for developing models of physical phenomena.

In my previous studies (Louca & Constantinou, 2002; Louca *et al*, 2003) SC was easy to learn and used for developing programs as models of physical phenomena. For instance in the first study, learners were able to recognize the usefulness of the software for modeling in science, while developing, testing and revising models of the linear propagation of light. In that study, students who were simply learning to use SC as part of a science club, identified SC as a tool for better representing a model of the linear propagation of light. SC is also accompanied by a tutorial, which consists of a sequence of interactive activities which I found to be successful in teaching students how to use SC in both previous studies.

I chose *Microworlds Logo* to be the textual-language programming software. During my pilot study (Louca *et al*, 2003) students easily learnt to use this software. Microworlds Logo is a version of Logo (Papert, 1980) designed for developing microworlds. It has the capability of graphical representations, and enabling objects' interactions. Programmable objects can be represented by actual pictures of the objects. Animation is easy to accomplish and the user can import graphics and sound. These features can make the outcome of the programming process to look similar with the outcome of the Stagecast Creator's programming process, which can be used for comparing different programs that result in the same outcomes in the different CPEs

Prior to the study, the school's County tested and approved Stagecast Creator and Microworlds Logo software to be used at the school's computer network by students. The company that produces Stagecast Creator software granted me 5 computer licenses of the software for the duration of the study. For the MW, I purchased 5 licenses for use at the school.

2.2.4. Purpose of the study

The central research question for this study is *how might computer-based programming environments support fifth grade student inquiry in physical science*? For this purpose, I have three subsidiary goals. (1) In investigating the (potential) role of CPEs in supporting thinking and learning in science at the elementary level, I describe how fifth graders use CPEs, and the process of programming to develop models of physical and/or biological phenomena. (2) To investigate student inquiry, I investigate how students use their prior knowledge, experiences and thinking strategies within the context of working with CPEs in science. (3) Finally, I provide descriptions of Microworlds Logo and Stagecast Creator, including descriptions of the characteristics of the process of programming that support student thinking at the fifth grade, in the context of developing models in science.

2.3. Study phases

The study was divided into two phases. The first phase was devoted to learning the program language and some modeling procedures and the second part was devoted to exploring ideas in science with the use of the programming environments. The data analysis for this study is based on the data that I collected during the second phase of the study.

2.3.1. Study phase one

The first phase of this study took place during the first 6 meetings (September 2002 through late October 2002). Its purpose was to teach students to use the programming environment to which they were assigned, and to introduce them to some modeling practices.

Stagecast Creator group

During the first sessions, students in the SC team familiarized themselves with the environment using the software tutorial that was previously demonstrated to be a successful tutor for this software (Louca & Constantinou, 2001; Louca *et al*, 2003). The tutorial is an interactive environment presenting the capabilities of the software (e.g. that you can have characters move, go into different backgrounds, have multiple rules etc) and showing the user how programming is done,(e.g. how creation of rules happens). The tutorial breaks up a desired microworld into small pieces, each of them being a potential rule, and guides students through the process of developing rules. At the beginning it shows them how the rules are created, then it guides them through the process of creating individual rules. The focus of the tutorial is on the process (steps) of creating rules that would assign characters with particular behaviors.

After going through the tutorial, I introduced students to several examples of ready-made microworlds in order to investigate their design, and practice their programming skills by altering features of the microworlds. These activities were based on the idea of asking students to figure out how the simulation is created, that is, what kinds of rules would create the visual results on the screen, and asking them to make changes in the simulation such as double the speed, change the direction, or add simple behaviors to the characters. This approach helps students to focus on the rules that create the simulation and to think of these as the mechanism of creating the simulation. Towards the end of this phase, I presented students with a task that required them to develop microworlds of their own, without providing them with any pre-programmed parts. In those cases I usually had a conversation with students in a whole team setting, in which they exchanged ideas about how to program this situation, what kinds of rules were needed and for what behavior.

Microworlds Logo group

Given the different type of programming language that MW uses, the focus of teaching students to use it was different from the focus of the SC tutorial. My teaching focus with the MW team was on teaching students programming primitives that are required for developing simple programs, and on basic program structure (e.g., a program starts with "to <program_name>, ends with "end", needs to have a "talkto <turtle's_name> etc).

Teaching, however, was done in a similar way with SC, by presenting students with several simple pre-programmed microworlds, asking students to figure out how the behavior of the characters was created and how to modify that behavior. These activities provided students the opportunity to investigate the capabilities of the programming environment, to develop an understanding about the function of programs in MW and familiarize themselves with the programming primitives. Towards the end of phase one, I asked students to develop their own simple programs. Students were allowed to use code from programs that they previously had seen or used. During that time, we usually had a conversation in a whole team setting, for exchanging ideas about what kinds of behaviors to assign to characters (such as, should the ball move with the same speed or should its speed change over time?) and for talking about how to write programs to represent and create those behaviors.

In both groups, students did not have the opportunity to learn every primitive or rule type. However, by the end of October, they had some sense of what it was like to create programs. Primitives or rule capabilities were presented later in the study too, when students had particular ideas about programming something, but did not know how to write them into programs (e.g., a sub-routine for running two programs at the same time in MW or a rule for giving priority for running a number of rules over all the others in SC).

2.3.2. Study phase two

During the second phase of this study, students worked on developing a model of a physical or biological phenomenon. Prior to any work, I asked students to decide what each group of 2or 3 students would like to program, as their final project. The only requirement was that their programs should be related with science. Each group in both teams had some time to think about their ideas and how they would program them. Students spent a meeting brainstorming ideas and looking through other students' programs related to science and mathematics (that are readily available on the websites of the companies that created the software of the study).

In proposing this study, my initial plan was to have students investigate a question framed around a physical phenomenon within students' experiences that would require children to explain what they think was going to happen and what would cause it to happen the way they thought. However, during the first phase of the study, when I presented students with similar (but smaller) tasks, some students were not as engaged as they were in other days of the study. For this reason, I decided to provide students with the ability to choose their own ideas to program, but provide them with criteria about choosing among their ideas. The criteria included that their programs should include or show a phenomenon from science, and that (for educational purposes) their programs should not include killings, or bombs, which apparently are very interesting topics for 5th graders! Therefore, some students working with SC, for instance, decided to create games that include physical phenomena (balloons with helium flying) and some others to create games using biological phenomena (ecosystems of fish in a small pond).

2.4. Data sources

Three primary sources of data were used in this study. First, videotaped *students' group work with SC and MW* that includes both their interactions with peers and myself. Second, *discussions* that I facilitated *in whole class* about the phenomena under study were videotaped and used as a primary source of data. Third, s*tudents' programs* (the outcomes of students' work with SC and MW) were also a primary source of data for this study.

2.4.1. Identification of groups that were videotaped

During weekly meetings, students' discussions within their groups, while working with computers were audiotaped. Two video cameras were also used for videotaping two of the groups during individual group work. For this reason, during phase one of the study, I chose the groups to be videotaped during the second phase of the study. My decision was based on the ability of students in the group to collaborate. Data sources and subsequent data analysis for this study focused on the conversations that students had during their work with computers. For this reason, it was important that I collected conversational data from teams that could collaborate, have conversations about their work and during their work.

For the SC team, Annie & Bryan (group) and Zen & Seth (group) seemed to be two focused groups that regularly participated in the club, and thus I chose to videotape them. Those students were also comfortable in communicating with one another in their groups while exchanging ideas about programming. Sean & Tyra (group) was the third group from which I used transcripts of their conversations during analysis (see description later) Similarly, Joe & Samir (group) and Aaron & Richard (group) were the two of the groups working with MW that I chose to videotape during the second phase of the study. These students were participating on a regular basis in the club and their group collaboration was usually adequate. Jiana & Gabriella (group) was the third group working with MW, from which I used transcribed conversations during analysis. The rest of the groups were simply audiotaped.

A possible concern is that the number of girls that was included in the groups that I used for analysis was low. Having groups with rich conversational data was my first priority, and in case of MW, Jiana and Gabriella were the only regularly attending girls for the team. An additional girl was not participating regularly. In the case of SC team, all-girls groups seem not to work very well, despite several combinations that I tried. Thus, I chose one group with one girl to be videotaped and used transcribed conversations from another group with one girl during analysis.

2.4.2. Reviewing video and audio data and transcribing

Reviewing video and audio data was done in two major phases. During the data collection period, after every session with each group, I watched all available video, which included any whole class discussions and the work of the two individual groups per software team. I did this in part for planning for next sessions, in order to review the individual group work, see what students had accomplished and prepare my teaching agenda and focus for the next session. While reviewing the videos I took notes about what was going on in the videos, with a focus on student thinking.

After the end of the collection period of the study, all videotaped conversations were transcribed, including conversations of the whole team and small group work. Transcripts of group conversations were analyzed using contextual inquiry (see section below for details) developing codes for *emerging patterns* of students' activities and conversations. For each software, students' work was analyzed separately. For initial analysis, I used transcripts of group work from the transcribed videotaped conversations (two groups per software team). Analysis revealed *emerging types* of students' activities and conversations, which consisted of particular combinations of emerging patterns, observed in both groups that I analyzed. At that time, I reviewed available audio from a third group from each software group, looking in particular for supportive evidence for the emerging types of students' activities and conversations (instances that those patterns occurred) or possible counter evidence (instances that in similar contexts such as typing a program, a different emerging type of students' activities or conversations could have occurred). Those instances were also transcribed and analyzed.

In reviewing and transcribing video and audio data, I excluded instances in which students were taking time off to discuss issues unrelated with the study (such as talk about their homework, events that happened during the school day). Those instances were noted in the data presentation as gaps in students' conversations.

2.4.3. Secondary data source

My journals during the data collection period of the study were used as a secondary data source. I kept daily journals before and after each session. Before each session I summarized my goals for the upcoming meeting with students and the activities that I planned to use, as well as the rationale behind them. The journals that I kept after the sessions had two parts. I wrote the first part of the journals immediately after the sessions, before reviewing and/or transcribing any conversation. In that part, I documented my experience and my observations from the meeting. For this purpose, I took approximately one hour after each session to describe in writing all the activities that I used during the meeting with the students, and how students responded to these activities. I also made logs of any piece of student inquiry that I observed or any interesting student comments and/or contribution during individual group work, or whole group discussions. I wrote the second part of my journals after I reviewed and/or transcribed segments of students' videotaped conversations, and possible student work with CPEs. In this part of my journal I documented instances of students' inquiry and particular use(s) of CPEs that I had identified from the videotapes. That was also a partial guide for organizing the next sessions' activities.

2.5. Data analysis

This case study research seeks to investigate and document students' work with two CPEs while developing models of physical phenomena. For analysis purposes, each CPE team was treated as a separate Case (a total number of two cases: SC and MW). The case units for this study were the different student groups working with CPEs. The subunits of this case study were the weekly meetings with the students. For this purpose, analysis and presentation of findings were based on three groups from each software group (case study units) following their work in detail for almost two months of meetings (case study sub-units). After analyzing and presenting the findings for each Case (software), I compared findings from the two different Cases, isolating their differences and similarities for students' activities and conversations. Then, I combined the findings from both Cases to discuss emerging themes about student thinking, student work with programming media and student computer-based modeling.

For the data analysis I used three different types of analysis: *contextual inquiry*, *analysis of student conversation*, and *artifact analysis* of the designed microworlds (programs). I used *contextual inquiry* in conjunction with *analysis of student conversation* in order to gain better insight in students' activity and conversation patterns while working with CPEs. Using the same source of data, (students' conversations while working with CPEs) I analyzed them from two different perspectives. Findings are results of the triangulation (Stake, 2000) between different sources of data and different ways of analyzing these sources. Below I provide detailed descriptions of the three types of analysis.

2.5.1. Contextual inquiry

I analyzed video and conversational data of children's work with CPEs using a modified version of *Contextual Inquiry* (Druin, *et al*, 1999). Contextual Inquiry is a method of collecting and analyzing data of children's activities and conversations, which can be used when children work with technology. Adults are usually part of the working group that is using technology in a specific situation. In my modified version of contextual inquiry, my role (as usually the only adult in the room) was minimized to the role of the facilitator: I was walking around the different groups and asking students clarification questions about their work, in addition to providing any help that students requested.

Contextual inquiry involves the analysis of student work in a particular macro-context such as using technology and computer media. However, contextual inquiry does not involve description or analysis of the micro-context[2] in which student activities take place (that is, particular situations that possibly students put themselves in while working with technology), because despite the fact that the context is important, the focus of contextual inquiry is on people's activities (Druin, *et al*, 1999) and communication. Analysis of student conversation and the accompanying narrative involve description of the particular micro-context of students' activities during their work with CPEs. For the purpose of this study, I used contextual inquiry for analyzing students' activities and conversations in small groups, while programming.

---

[2] With macro-context I mean the particular environment that student learning takes place, whereas with micro-context I refer to particular student activities within the learning environment. An example that is relevant with this study is MW as the macro-context, and the activity of debugging that puts students in a mode of seeing the code as causing a phenomenon, as the micro-context

After videotaped conversations were transcribed, transcripts were transferred in a cell-based diagram (matrix) with three columns for analysis, as indicated in Figure 2.1. The first column included the transcript of the conversation. I placed every student utterance in a different cell (decision adopted from a study investigating student interactions while working with SC (Underwood, *et al*, 1996)). In that study, researchers were investigating whether students' work in small groups with CPEs was related to their performance. For analysis, they used an interaction process analysis (Bales Interaction Process Analysis) which required that each utterance of student conversation was classified into one of the categories used for the analysis.

During contextual inquiry, moments of silence were represented in separate cells, and were added after reviewing the video. In the second column I coded students' activities during each utterance, after reviewing in detail the screen-capture videos. In the third column I coded the type of the conversation that students had. In the case they were having a conversation, mostly about what they were talking about and how, following the analysis of the pilot study (partly reported in Louca, *et al*, 2003). In cases that students were silent, I coded whether they were just silent or whether the rest of the students were not at their group.

Transcript	Activity	Conversation

Figure 2.1. Example of the matrix used for contextual inquiry

Development of codes

Codes for students' activities and conversations were, (1) directly adopted from my pilot study without any modification, (2) adopted from my pilot study and then refined into more detailed categories, or (3) developed as new categories.

Codes used for activity and conversation patterns were partly adopted from my pilot study (Louca, *et al*, 2003), that investigated similar issues with students using SC and MW for developing models in science. However, that study was a small scale descriptive study, and findings were not intended to be highly detailed. Some categories for coding in that study were more general, and some of the findings were not very detailed, which was an indication for the need for developing either subcategories or more detailed categories. For instance, "programming" was a category coding activities that included the activities of both typing new code and correcting existing code. However, findings did not reflect the difference of typing new code or correcting existing code (which might be considered as possible sub-categories). Thus, I decided to create more categories that would be more descriptive. For instance, categories a2, a3, a4, a5 (see Table 2.1) were developed using the findings from the pilot study. Other categories such as b1, b2, b8 were directly adopted from that study without any modifications.

Several categories were created during the coding process of the transcripts. For this purpose I adopted an analytic procedure from grounded theory for developing new categories: the process of open coding (Strauss & Corbin, 1998). I began coding of a transcript with the categories that I transferred or developed from the pilot study, and when I reached an utterance that would not fit into one of those categories, I developed a new code. I continued this process for all the sessions of one group per software (usually

4-5 meetings). After that, I reviewed the categories, made decisions to drop (or group) categories that were not often in the transcript and that could be summarized under a different category (e.g., a9, and b10). Then I reviewed once again the coded transcript, making sure that codes were reflecting my final decisions. During coding the rest of the data however, there were instances where I had to redefine the meanings of several codes, and in some cases add a few more codes of activity or conversation patterns. Therefore, after finishing coding all the transcripts, I double checked all the transcripts, making sure that the codes assigned to each utterance were consistent with the final list of codes.

At that point, I gave all transcripts and a list of all the codes with their definitions and one or two examples to a second coder, who went over and re-coded the transcripts. She did not have access to my analysis (the codes that I assigned to each utterance). The second coder was partly involved in the study by visiting on Wednesdays and helping with the club logistics (setting up equipment, helping student groups with their work etc.). After she finished coding all the transcripts, we both went through the transcripts comparing the codes that we had assigned to each utterance. When we isolated differences in codes, we talked about them and resolved the differences. The second coder has a background in quantitative research methodology and with teaching mathematics and science at the elementary level, which provided her with a critical view of my assignment of codes, in the cases of disagreements.

Table 2.1 provides a summary of the categories that I used for coding students' activities and conversations, with examples from coded transcripts for each category. Several categories are descriptive of students' activities (such as write code), some of which happened during moments of silence, and thus are not accompanied by examples.

Categories B8 (describe their program (what they did already)) and B9 (describe what to program in future) are listed and have been used as categorizations for student activities because, when a utterance was coded with one of the two codes, students were not working on their computers, but were simply talking about what they had programmed or what they were planning to program. Their conversations were further coded for the kind of conversation: whether students talked about depicting details in their program (C1), about the overall story of what they were designing (C4), whether they used the program language to talk about their program (C3), or about the program language itself (C4).

Activities	Examples of coded utterance
A1-read code from screen	No examples are provided for these categories, because they were used to code solely activities that were not necessarily accompanied by any conversation. Videos of students' work was primarily used for assignment of these codes
A2-type code from scratch	
A3-correct code for depiction	
A4-correct code for science	
A5-correct code (the program was not working)	
A6-correct code to fit their story	
A7-delete code	
A8-create variables	
A9-other logistical issues (save their programs etc)	
A10-whole line inaudible	
A11-debug code sequence	
A12-gap	
B1-run the program	
B2-read feedback from the computer	Richard: <reading from feedback> I don't know how to a in shoot.
B3-get/deal with characters of their programs	Joe: we need to get a target as well.
B5-get/deal with backgrounds of their programs[3]	Seth: we need like a map [for background], like a cross section.
B6-make buttons for their programs	Samir: we need to name this [button]! <while typing> Arrow.
B7-explore the program windows/capabilities	Zen: please, please, please, I'll, I'll, I'll just gonna… Let me see. Ok. There aren't any stages. Now, are there any specials?
B8-describe their program (what they did already)	Samir: […] So I had to make this [amount of forward] a bit higher for do like this <gestures>. I just didn't seem to like this. SO I changed it to <inaudible>. The angles, it's going up Zen: so this guy knows how to walk right. He walks right, he walks 2 spaces…

[3] Category B4 was left out by mistake and was never used as a code.

B9-describe what to program	Joe: well, we would probably like, we probably need to put at the beginning right, in front of that, so that the angle changes, because if we start with the fd and and then if you do the right, it would go like this Seth: we have to make his energy speed start at 30 or something.
B10-interruption	Seth: what's wrong? It's [is the rule] first? Zen: No, see it's adding that's why. See!
B11-talk to find bugs in their programs	Samir: that was way too many repeats [for right 0.5]. Joe: I know what we have to do, we have to do seth 90.
B12-contol-F (switch programming and simulation windows, applicable only for MW)	
Conversation (talk about…)	
C1-depiction	Zen: … whow! It's [this character] huge! Seth: Do they [the characters] have different appearances?
C2-what to program/the program itself	Richard: um, we just need to make something that does listen to the sliders instead to the fds. So it goes fd the amount of the slider is on. Seth: no, we have to make a new one, we have to make a new rule.
C3-how to program/ what code to write	Aaron: yea make it a bit more ….the first one is fd 2…2 wait 4. So we need to double all of that. Zen: … our bottle 1 is gonna be full of Gatorade. And that's gonna depend, depending on how much Gatorade you have in your runner is how fast they go.
C4-about their story (not referring to code)	Aaron: 'cause there's less, see <showing the simulation>, this [the astronaut on the moon] goes up and way, way higher… [then the boy jumping on the earth] Zen: …ok, one [character] is bigger … but has to slow down a lot.
C5-silence	
C6-one leaves the group/does not pay attention	
C7-"click here"	Zen: […] hold on let me show you just need to… It's so easy. No, no, no, don't do that, don't do that. You just need to <takes the mouse and shows on the screen>
C9-make references to experiences	Seth: yea, it's just like, I think it kind of make sense that 0 would mean, like when you are running really fast, sometimes when you chasing someone you don't stop running, stop during and starting slowing down.
C10-"how do we do that?"	Richard: ok, we have no clue how to make the jump moon together, see look what happened. How do we do that?
C11-"now, I click here!" (talk their way through programming)	Richard: I have an idea <starts changing the numbers in the program> I ….2, 3, 4, 5, 6 and then…
C12-"-what are you doing? -something!"	Aaron: I have an idea, I have an idea. Get off [the keyboard], get off. Aaron: what are you doing? Richard: wait, wait I am doing something, guy…

Table 2.1. Categories used in the contextual inquiry

Separated coding for activities and conversations

Contextual inquiry does not necessarily separate students' activities and conversations. Findings from the pilot study (Louca *et al*, 2003) comparing the communication within small groups of students, showed differences in the activity and communication patterns between students working with SC and students working with MW. Thus, it was important to steer this investigation towards those differences, making sure that additional possible differences would be captured by analysis. This required separating coding for student conversation and for student activities.

Coding activities and conversation separately was one of the first issues that came up during coding of the conversational data in the study. As the following example indicates, 10 utterances of conversation might have been coded with the same activity pattern but different conversation pattern and vice versa.

	Transcript	Activity	Conversation
1.	**Joe:** the problem with this is that we have to make the arrow go like that <showing on the screen the motion of the arrow>	b9	c4
2.	**Samir:** let’s make <inaudible>	b9	
3.	**Joe:** like that <showing the first half of the motion to the top of the trajectory> and then when it turns it goes like uuuuuuuuuuu <gestures showing the direction of the arrow changing from going up to going down>	b9	c4
4.	<silence while clicking/typing/looking on the screen>		c5
5.	**Joe:** see if you try to program from <inaudible> you need to program this part first, separately from the <inaudible> going like this.	b9	c2
6.	<silence while clicking/typing/looking on the screen>		c5
7.	**Joe:** we need to get a target as well.	b3	c4
8.	**Samir:** we need to name this! Arrow	b6	c11
9.	Control - F	b12	
10.	**Samir:** to arrow. Um, I will call it…..	a2	c11
11.	interruption	b10	
12.	**Samir:** talk to arrow… what is it called? It is called t, t what?	a2	c3
13.	**Joe:** yes it's called t1, because	a2	c3
14.	**Samir:** not it's called t3	b9	c3
15.	**Joe:** no, it is t1 because we haven't made programs the other 2 ones	b9	c3

In lines 1 through 5 students were talking about their plans for programming (b9 refers to “talking about what to program”). For lines 1 and 3 Joe was not using any program language while describing his ideas. In line 5, however, he stared talking about the kinds of programs that they needed to create, talking about having separate programs for probably particular parts of the phenomenon (which is coded as c2: “talking about the program/code”). After that there is a brief moment of silence and then the next nine utterances are coded with seven different categories of activities, but the conversation is only coded with two categories (c11: “talking their way through programming” and c3: “talk about how to program”). In other words, even in cases where students’ activities were coded with the same category, their conversations were not necessarily the same, and in cases where student conversations were the same, their activities were not necessarily similar. This supported my decision of coding activities and conversations separately.

Findings derived from analysis with separately-coded activities and conversations, have the possible advantage of being independent from each other (no *a priori* decision was made about developing or adopting codes for analysis that would account for both activities and conversations at the same time). This can support later any efforts to explain or relate findings about conversations with findings about activities, grounding such relation in the data themselves. In this way, any possible relations between activities and conversations are not caused by the analysis, since the data were analyzed separately with respect to conversations and activities.

Lastly, as I indicate below under “development of codes”, some codes that were developed during analysis in the pilot study were general and were not differentiating between sub-categories. This was another indication for the need to have more analytical codes, supporting the idea of having separate codes for activities and conversations.

Presentation of findings

Students' activities and conversations patterns are presented in time-line graphs, following the approach that Schoenfeld (1989) has used in presenting findings from analysis of student work on problem solving in mathematics education. I transferred students' activity and conversation patterns on graphs (see chapters 3 and 4 for analytic presentation and summary of findings), presenting separately activities and conversations that students in the study used while working with CPEs. I analyzed four student groups (that is, four units), two from each software team (that is, two cases). For each student group (unit of the cases) two graphs were produced. Instead of using time on the x-axis of the graphs, I used the numbered utterances, because the purpose of the presentations was to reveal particular combinations of the activity and conversation patterns.

Emerging types of activities and conversations

After transferring all data on graphs, graphs from units of the same Case (groups from the same software team) were compared to isolate any similarities in the combinations of the patterns of students' activities and conversations. All combinations that were similar among all 3 analyzed groups for each software, were then combined with the analysis of conversation (see description below) to link combinations of activity or conversation patterns with the particular (micro)context in which they happened and with students' purpose during that part of their work. From this combination of contextual inquiry and analysis of student conversation, activity and conversation types emerged.

Activity and conversation types are specific combinations of student’s work or conversations, that were identified from graphing patterns of students’ work and conversations in graphs of patterns vs. utterance. In this sense, these types of activities or conversations emerged from the data, and are descriptive of the kind of work or conversations in which students were involved. The act of coding a student conversation, for instance, is used to provide some analytic perspective of what students are saying in that conversation. Because of the nature of the activity of coding to break down the conversation into small pieces to be coded, codes and coded transcript can only give us limited information such as the number of total times that students used particular conversation patterns. Graphing, however, coded transcript with respect to time (or the sequence in which they happen), can provide an overview of the students’ conversation patterns which can possibly lead into grouping these based on groups with similar characteristics. Thus, activity types, for instance, are combinations of students’ activity patterns during particular parts of their work, which unless stated otherwise were found in all groups more than once. In this way, findings from contextual inquiry (patterns and types of activity and conversations) can be combined with analysis of conversation mostly for the purpose of linking activity and conversation types with the (micro-) context in which they occurred (see description of emerging themes below).

2.5.2. Analysis of student conversation

Analysis of student conversation was the second type of analysis that I used in this study. It is a multidisciplinary approach of analyzing text (in the case of this study, transcribed student conversation) as a gateway to student thinking and experience. Patterns of students’ conversations (revealed by contextual inquiry) and analysis of student conversation are presented together to triangulate findings. Analysis of student conversation provides in detail the particular context in which students' work (activities and conversations) took place, the content of the conversation and possible relations between the content and the context of the conversation. Finally, analysis of student conversation seeks to identify possible student thinking patterns that are evident in the conversation.

Mapping relations between content and context

Analysis of conversation seeks to describe the context in which students' conversations took place, situate the activity or conversation types that were observed in that context and describe students' conversation in those details that were possibly lost from contextual inquiry. In contextual inquiry, analysis was based on coding for students' activities and conversations. However, the conversations, for example, happened in a variety of situations (micro-contexts) that contextual inquiry does not account for, such as while students were away from the computers, while programming, while debugging, while changing how their simulation looked etc. Therefore, the purpose of the analysis of student conversation was to map possible relations between the conversations and the context in which they happened. In this sense, presenting a description of a conversation includes what was said, how this was said, for what possible reason this was said and in what particular context.

Analysis of student conversation is multidisciplinary because it uses research techniques and approaches originated from linguistics, educational psychology and educational research (Edwards & Mercer, 1995). The analysis of student conversation also follows examples of such approaches for analyzing student conversation in science and mathematics (e.g. Ball, 1993; Gallas, 1995). In their work, Ball (1993) and Gallas (1995) analyzed student thinking in mathematics and science, using the communication among students and the teacher in regular classrooms. They used transcripts of students' conversations to isolate students' ideas for the topic under investigation and tried to identify students' reasoning behind these ideas and the different ways students articulate the ideas.

In the analysis of student conversation that I used in this study, I also aimed to isolate, describe and analyze instances where students have conversations, use CPEs' tools and use nascent abilities they have that can be supportive for modeling in science. Possible examples are the ability to provide descriptions of the underlying mechanism of the physical phenomena, the ability to make references to experiences, the ability to make distinctions between different types of knowledge, and the ability to use code (programs, rather than simulations) as representations of physical phenomena.

Discourse as the primary source of data

Analysis of student conversation is different from discourse analysis (Sinclair & Coulthard, 1975), which is an analytic approach in linguistics focused on exploring how language (in text format) is organized in units larger than the sentence (Edwards & Mercer, 1995). The purpose of discourse analysis is to analyze text in a way that would reveal the structure of what is said rather than the content of what is said.

Even though discourse analysis and analysis of conversation use the same type of data, analysis of conversation does not seek to reveal the structure of the talk. It is rather focused on the context in which the conversation takes place and in the content of the conversation (Edwards & Mercer, 1995). For this reason, in analysis of student conversation I did not follow a process of coding text with particular codes (I did that for contextual inquiry), but I provide detailed descriptions and possible interpretations of the conversation, that are meant to be read in parallel with the transcript. In this sense, I follow the research approach of educational research (Edwards & Mercer, 1995), which seeks to develop a sense of what takes place in the classroom in an effort to map possible relations between the learning processes and the discourse.

Mapping relations between talk (discourse) and student thinking

In describing what is said and how it is said, I also seek to build interpretations about students' thinking abilities in the conversation. This approach shares some similarities with psychological approaches for research in educational settings (Edwards & Mercer, 1995). Research on classroom interactions have been mostly concerned with the identification and modification of patterns of activities by, for example, the use of positive or negative reinforcement (Edwards & Mercer, 1995). More generally, research approaches in educational psychology have focused primarily on the identification, study, and measurement of abilities and behaviors of individuals rather than the origin and ways of using those abilities. Often, research in psychology that is less individualistic has a developmental approach for explaining the use of abilities in the student population.

My approach for analyzing conversation shares the interest of educational psychology research, which investigates the abilities of student population. In my approach, however, I do not use a developmental approach for studying student abilities. Nor am I focused on the study and modification of behaviors. Analysis of student conversation that I used in this study uses transcribed conversations (discourse) as the data source. This is another difference with psychological research because discourse, is not usually its focus. The purpose of analysis of student conversation was to identify thinking and working abilities that students use in their conversation, and describe them through the particular conversation (context). For this purpose, I provide raw transcript data of students' conversations, and possible explanations of what kind of thinking is taking place in the conversation, as a possible gateway in student thinking.

Analysis of text (conversations) in science education

A current emphasis in science education research is the use of discourse as a primary source of data. This has been the approach for a variety of studies: Kurth, et al (2002) investigated the narrative and paradigmatic expression in elementary science discourse, van Zee et al (2001), reported case studies documenting and interpreting student and teacher questions during science discourse and van Zee (2000) analyzed student-generated inquiry discussions. Hogan, Natasi & Pressley (2000) examined discourse components, interaction patterns and reasoning in open-ended student conversations with their peers about science, focusing on "the nature and sophistication of collaborative scientific reasoning with and without teacher guidance" (Hogan, Natasi & Pressley, 2000, p.380), stressing the importance of research in naturalistic settings (rather than research in laboratory settings). Their analysis captured interactive protocols among students in small groups, which are records of purposeful conversation events in small groups while working. The student conversations in this study were mostly about experimental data that students had collected.

van Zee (2000) provides a comprehensive description of research in science education that has used classroom and/or small group discourse as primary data sources analyzed in a variety of ways is provided by. These different ways of discourse analysis in science education include argument analysis (Toulmin, 1958), analysis of the development of common knowledge (Edwards & Mercer, 1987), examination of types of warrants that students used to justify claims based on collected data (Kelly et al, 1998), tracing conceptual change as an interactive social process (Rochelle, 1992) and analysis of “reflective discourse” (van Zee & Minstrell, 1997).

Research in science education, however, has not widely used analysis of students’ *theoretical conversations* (that is conversations about natural phenomena without any experimental data available, like conversations that Gallas (1995) provided and analyzed), because, traditionally, the focus has been on experimental science. Children, like scientists, (should learn to) design experiments, collect and analyze data, and develop or modify ideas (theories) based on the experimental results. Research in science education has focused on students’ abilities to design, carry out experiments and interpret results and the development of those abilities through science education, following recommendations of the NSES (1990), which place emphases in early grades on science as empirical inquiry, to help students develop abilities for observation, experimentation, forming conclusions based on evidence they obtain through experimentation, and on the logico-mathematical abilities for controlling variables and organizing data.

The focus of the analysis of student conversation in this study was on theoretical conversations that students had while trying to represent physical phenomena with CPEs. During this study, students did not have access to any empirical evidence, and thus they were engaged in conversations about natural phenomena, during which they turned to experiences that they had from their everyday lives. That is, without any empirical evidence available, students can still make progress by e.g., referring to everyday experiences. This was one of the purposes of this study: to study what kind of conversations students would have during developing representations of natural phenomena, without any experimentation involved.

2.5.3. Artifact analysis for the designed microworlds

For the purposes of the artifact analysis, I reviewed students' programs. For the second phase of the study, each group developed several models of particular phenomena, each revised from a previous one. Findings from contextual inquiry and analysis of student conversation guided the process of analyzing student's programs/models. Artifact analysis focused on two issues: the differences of sequential programs that students developed during the study and on the particular ways that students represented phenomena.

During their work with CPEs students develop a number of programs, during iterations of writing programs, running them, talking about their programs or their simulations) and then making revisions. Partly, artifact analysis was used to identify the modifications of what students did in their programs, relating those with findings from contextual inquiry and analysis of student conversations. For this purpose, I did not present findings from artifact analysis separately, but rather I refer to students' program characteristics as support to arguments about their work with CPEs.

Artifact analysis also investigated the kinds of representations that students used in the study to represent physical phenomena with CPEs. For this purpose I investigated issues related to whether students' programs were used to create a simulation, simple descriptions of the phenomenon or constructed models of natural phenomena, whether programs included relations between physical values represented in the programs or not and whether programs represented the mechanism that causes the phenomena.

2.5.4. Use of secondary data source: my journals

I used *my journals* in conjunction with the other data to support the patterns that were revealed. Mainly, my journals were logs of the meetings I had with students during the study (data collection period). I made entries about episodes of students' conversations or work with computers that could have been modeling conversations and activities or triggered these. During data analysis, I used my notes as a guide to episodes of students' work and conversations with CPEs that I analyzed using analysis of student conversations.

2.6. Researcher's role & ethical issues

2.6.1. Researcher's role

I was the primary researcher for this study and also the teacher of the science/computer club that was the source of the data for this study. During the study, I was playing both roles, although it was important to have them separated. During the data collection period, when preparing for the daily session and while working with children I was undertaking the role of the teacher of the club. At the end of each session and usually the day that followed, I was undertaking the role of researcher, watching and transcribing videos and reflecting on the sessions. After the end of the data collection period, I took the role of the researcher, transcribed all the materials analyzed in this study and analyzed them.

Separating the two roles (teacher and researcher) was important partly because in this study my goal was not to study my personal teaching practices. My purpose was to investigate how students used the study's CPEs to develop models of physical phenomena. However, I found that the two roles often contradicted one another. As a teacher I followed an approach in which students are provided with the freedom to work in an open-ended learning environment. In this way, it is possible to study different ways that students are inclined to use CPEs, without the teacher prompting those uses.

On the other hand, however, I was also struggling with my role as a researcher who expected to see a number of particular uses of CPEs from students that are productive for science. In fact in the first two months of the study, I sometimes "pushed" students in a direction which might have been toward what I thought of being productive, but at the same time I was not giving students the chance to find their own productive ways for thinking about science.

At some point prior to phase two of the study, I decided that I could not separate enough the role of the teacher and the role of the researcher and I stopped working on researcher tasks such as transcribing, analyzing student thinking etc. My focus was then solely on teaching and providing students with enough "freedom" to use their abilities for productive thinking in science.

In a way, it seemed that the conflict that I experienced between the two roles (teacher and researcher) was a conflict in my conceptualization of the two roles. "Pushing" students into a particular direction (as a teacher) because (as a researcher) I wanted to start "doing some modeling" so that I have enough data for the study, was aligned with a view of a teacher focused on where he wants students to be rather than assessing deeply where they are and providing the appropriate learning environment and experiences to help them move on. Most probably the conflict was on the ways in which my role of researcher interfered with my views of teaching.

This unique experience as a teacher and as the primary researcher in the study, as well as the combination of those two roles, changed my view of teaching science to young children. I have previously worked as "researcher" in other studies with similar approaches, but since my first year of teaching 5th graders (back in 1999) I did not have the experience of teaching. One of the contributions of this experience involved my views for scaffolding in teaching and learning. In one view that has been demonstrated to be important in this study, effective teaching is providing learners with the appropriate learning environment. The teacher's role is partly to provide that appropriate context where productive science conversations and modeling in science can happen. The descriptions that I provide for findings later on in this report are in one way an effort to capture what seemed to be productive scaffolding for students and possibly to explore how far that scaffolding should go.

2.6.2. Risks for participants

Students participating in the study may appear in segments of videotape that are available as a collection of case studies for this dissertation study and in the future as published case studies. It is possible that students might be recognized by someone using or reviewing these materials. For this purpose, students' full names were not used at any time during the study, and any reference to them in this report or any subsequent publication was and will be by their first name only. All other information about students (excluding their first name) is confidential and will not be identified at any time.

2.6.3. Confidentiality

All the students' work with computers and their discussions in whole class settings were videotaped. Children's work on the computer applications were also collected as part of the data. For the duration of the study, all videotapes and audiotapes were kept in locked storage at the University of Maryland. Students' saved work with computers were kept in secure accounts on the school's computer network that were created for the purpose of this study, following the security protocols of the school's county for students' work with computers. Backups of students' computer work were also kept in a locked storage at the University of Maryland.

After the end of the study, the students' work from the school's computers was removed and stored with the videotapes and the students' journals in locked storage at the University of Maryland. The data of this study will be destroyed by erasing the videotapes and the computer disks of students' work after the data are no longer of any use for this study.

2.7. Limitations

Limitations of this study include (1) the small number of participants and (2) the short duration of the study. Also (3) gender differences were not the focus of the study, (4) during analysis unfocused work was omitted from data, and (5) patterns of activities and conversations did not reflect the time that students spent on each activity.

This study was a small scale descriptive study, aiming to provide some descriptions of how students use CPEs as tools for developing models of natural phenomena. Students who participated in the study were divided into two teams (based on their preference for the day of their participation). For each team, 15 students were selected to participate, but only 9 remained in the MW team until the end of the study and 10 in the SC team. Thus, the small number of the students and the short duration of the study (from September 2002 until December 2002) are limitations for the study's claims. Of course, my presentation of the findings from this study had a "tone of promising possibilities" of using CPEs in particular ways with students, because findings were isolated from a total of 19 fifth-graders who were involved in the study.

An additional limitation for this study was the fact that it had the form of an afternoon science/computer club. This had several implications for the study. First, students did not view their participation in the club in a similar way that they viewed their participation in their morning classes. The study was for them an afternoon activity, and in several occasions were acting in this way, adding difficulties to my role as the teacher to use activities and topics of investigation that would keep them focused and excited. Second, students in the study did not see me as a regular teacher of the club, and in some cases 2 particular students were calling me by my first name only. This also added the difficulty of managing this group of students, keeping them focused on topics that included investigations of natural phenomena, and at the same time provide them with enough "room" to work in their own ways (that was one of the purposes of this study, to study how students use CPEs).

Even if there was not intentional effort to exclude any gender group from the study and the focus groups, investigation of the gender differences was not the focus of this study. Because of that, and given the priority in putting students in groups of two, in which they could collaborate and communicate, in some cases the groups were of mixed gender and in other were of the same gender. In choosing groups for the analysis, in both SC and MW teams, I included at least one group with one or two girls.

During analysis, unfocused students' work was omitted. Therefore, conversations that were analyzed with contextual inquiry and analysis of student conversations did not take into consideration rather large amounts of conversations in which students did not work with CPEs but talked or dealt with other issues related to their homework, their school day etc.

Patterns of activities and conversations that are presented in the figures in chapters 3 and 4 do not reflect the time that students spent on a particular activity because coding was done for each student utterance. For instance, moments of silence were usually longer than other codes used during analysis. For this reason, claims about emerging themes are strictly descriptive of student's work, in the sense of showing how work and conversations looked like in the study, while students were developing models of natural phenomena.

3. MICROWORLDS LOGO FINDINGS: CONTEXTUAL INQUIRY & ANALYSIS OF STUDENT CONVERSATIONS

In this section I present and discuss findings from contextual analysis of students' work (activity patterns) and conversations (conversation patterns) while working with Microworlds Logo. The presented findings were common among the three groups that were selected and analyzed for this study. Except when noted, similar types of activities and conversations were identified in all three groups that were analyzed in detail.

I start with a discussion about the differences between the activity and conversation patterns and types. Then I turn to a discussion about the different possible ways that I could have presented the findings of this study accompanied by a description of the challenge that I faced for deciding which way of presenting findings would have been the most appropriate for this study. Then, in two subsequent sections, I present the different types of conversations and activities of students' work with MW. Conversation types (CT) include CT I: Talking about their program's structure, CT II: Talking about program details, CT III: Talking while programming, CT IV: Talking about how the simulation looks, and CT V: Talking about what happens in the simulation. Activity types (program strategies (PI)) include a description of how students dealt with characters and backgrounds of their designs, PI I: Writing & debugging new code, PI II: Correcting depiction and PI III: Modifying code to change the science that represented.

3.1. Clarifications of terms and presentation of findings

For presentation purposes, I need to clarify three terms that I will be using. First, *activity and conversation patterns* refer to the codes of students' work and conversations that were developed during analysis, using grounded theory techniques (as discussed in the methodology chapter). Activity patterns for instance are students' activities (such as reading code, deleting code, changing code etc.) that were repeatedly observed. All these patterns emerged from the data.

Second, *activity and conversation types* refer to specific combinations of students' work or conversations, that were identified from graphing patterns of students' work and conversations in graphs of patterns vs. time/utterance. These types of activities or conversations also emerged from the data and are descriptive of the kind of work or conversations in which students were involved. coding student conversations, for instance, is used to provide some analytic perspective of what students are saying in that conversation. Because of the nature of the activity of coding to break down the conversation into small pieces to be coded, codes and coded transcript can only give us information about aspects such as the number of total times that students used a particular conversation pattern. Graphing, however, coded transcript with respect to time can provide an overview of the students' conversation patterns, which can possibly lead into grouping these based in groups with similar characteristics. Thus, activity types, for instance, are combinations of students' activity patterns during particular parts of their work, which unless stated otherwise were found in all groups more than once. In this way, findings from contextual inquiry (patterns and types of activity and conversations) can be combined with analysis of conversation mostly for the purpose of linking activity and conversation types with the context in which they occurred (see description of emerging themes below).

Lastly, *emerging themes* refer to possible emerging relations among different activity or conversation types that were revealed by the study's findings. Emerging themes are also supported by findings from analysis of conversations, in particular by descriptions of the context in which related types of work and conversations took place. For instance, working to fix depicting details in a program, and working to fix the science represented by a program are two different activity types (which are combinations of particular activity patterns). An emerging theme of possible *student abilities to change (shift) their working focus* was identified when students in the study repeatedly moved from the first type of activity to the second, shifting radically their work and conversation focus, based on the need of the particular context of their work.

In presenting activity and conversation types, I provide graphs that summarize student's actions and/or conversations during their work with computers (contextual inquiry findings) accompanied by episodes of students' work and conversations (analysis of student conversations). To support activity and conversation types, findings from contextual inquiry are coupled with discussion of findings from analysis of student conversations. In these cases, findings from both analyses were derived from the same data. Contextual inquiry provides a summary of the students' work and conversation patterns that make up the activity and conversation types, whereas analysis of student conversations provides details for the micro-context in which those patterns occurred with examples of student conversation and work. For example, a working pattern that I provide later consists of students going back and forth, from the programming window (where they type code) to the simulation window (where they run their program) and vice versa (pattern provided by the contextual inquiry). To fully understand this activity pattern, one needs to see in what particular micro-context were students doing this: in this case they were debugging, by making small changes to their code, trying them out, making some more changes and trying them out. This micro-context is provided by the analysis of student conversations.

Emerging types of students' activities and conversations that I present in contextual analysis fall under two major categories: student actions during their work with computers and student conversations before, during and after their work with computers. Presentation and discussion of the emerging themes follow that categorization in that order.

Students' activity and conversation types are related in two important ways. Firstly, some activities were observed in parallel with some conversation types. For instance program strategies that I present (activity types) are accompanied by conversation types (conversations during programming) and together they fully describe students' work with computers. Secondly, in different parts of their work, students may have been using the same conversation type with a different activity type, or similar program strategies with different conversation types. As I discuss in further detail later, I believe that this supports and justifies a separate presentation and analysis of activity and conversation types, showing that they might not simply depend on each other. In addition, and pending further investigation, these data might support a hypothesis that suggests that *students have a collection of nascent abilities of scientific inquiry that they can activate in different situations, based on the situational needs*. For example in one situation students might “see” that it is appropriate to use a particular program strategy with a particular conversation type whereas in a similar situation they use a different program strategy with the same conversation type.

In presenting the different types of activities and conversation, I across the challenge of choosing a presentation order. This challenge was an important one, because it was based on the key characteristics of the activity and conversation types: Should they be presented 1) based on the sequence of “appearance” in student work, 2) based on those characteristics that they share, or 3) based on the type of work during which they occur?

3.1.1. Sequence of appearance

Most of the activity and conversation types occurred in subsequent time periods, as successive events, e.g., students first set up the characters of their designs, then they wrote code, they debugged their programs etc. Presenting findings in this way may provide a better and possibly richer description of students’ work, because it simply follows them in time. However, this kind of presentation may imply that each different group of activities (e.g., debugging) depends on a previous one, possibly not focusing on other factors that actually cause that sequence of events.

3.1.2. Shared characteristics

A second possible way of presenting findings is based on their shared characteristics. In this way, I could group types of program strategies by those characteristics they share, even though they might have occurred in different times and possibly in different parts of students’ work. For example, consider two episodes of students’ conversation that occurred immediately after students finished typing a program: in the first situation students presented their program to a student from another group and in the second situation students debated on whether their program represented the phenomenon under study accurately. Even though both conversations occurred after some programming, the context and the focus of the conversations were totally different. In fact, as I discuss later, the two episodes were radically different from each other in the conversation patterns that analysis revealed. In addition, presentation of findings based on shared characteristics may fail to highlight small differences that are important to students' work, or overemphasize differences that are not important for using CPEs as modeling tools in science.

3.1.3. Shared context

A third way of presenting findings is based on the context of each type. Each type would be presented individually, supported by examples in the context in which it occurred. In this sense, activity and conversation types can be grouped in a variety of ways, even the ones aforementioned. In addition, findings can also be presented in several non-traditional ways, as the one supported by the hypothesis that I have developed during the analysis of data: *do students in the study use a variety of thinking and program strategies/resources depending on the context of each situation? And if so, what is the context of such situations?*

Presenting activity and conversation types independently of each other and within the context they occurred provides an additional advantage, this time methodological: the activity and conversation types can be used as starting points of emerging themes that can explain findings and possibly combine them in a united theory. Even though I do not claim that I am developing a theory (given the particular methodology that I use, the small scale and the short duration of the study), it is important that I present the findings in ways that future research can use them to design studies that can further investigate and possibly unite findings and emerging themes into a theory. Therefore, I have decided to follow this third type of presentation.

The presentation of students' activity and conversation types below is a description of the characteristics of students' work with CPEs. In this sense, it is a discussion of students' tendencies and abilities of use MW and SC in particular ways that may or may not be supportive for modeling in science. An important part of this presentation is whether students can switch focus in their work, and while using a particular type of conversation or activity, start using another. I have deferred this discussion for later, when I discuss shifts in students' work and conversations that occurred due to shifts in student focus. I have restricted the presentation of findings below to the characteristics of student work in different contexts, in an effort to highlight the fact that in the context of using CPEs to develop models of natural phenomena, students can use a variety of activities and conversation types.

Another important decision that I had to make was related to naming different types of students' conversations and activities. I was initially inclined to name the different types of conversations with the particular "mode of student engagement" because it was providing some information about the context in which the conversation or activity types took place. For instance, the CT I occurred in the context of presenting ideas to be programmed, and naming that way (e.g., present ideas to be programmed) supports my purpose to describe these types of conversation and activities in detail, including the context in which they occurred. On the other hand, however, different conversations and activity types can be named based on their characteristics. I decided to try to follow the second way of naming conversation and activity types following their characteristics, even though it was adding the difficulty of having to find an appropriate “name” that would represent the different activity and conversation types. In a few cases, I used the context in which the conversation/activity happened (e.g., conversations while programming) because it highlighted the importance of the context in which the activities or conversations occurred.

3.2. Types of conversations

Contextual analysis of student conversations revealed several different types of conversations that were commonly found in all three groups that were analyzed in detail. In the case of MW, the different types of conversations that I discuss below seem not to appear in any kind of hierarchy: for instance the first type wouldn’t necessarily be found before the second type of conversation. Rather, students seemed to be able jump from one type to another according to their conversation needs. A possible assertion is that *given the different needs for communication, students use different parts of their program (the simulation, the code that created the simulation) to fit those needs*. This brings up two interesting points: (i) students can navigate between those conversation types and adjust them accordingly to their needs; and (ii) MW has a variety of different tools that students can use in different situations.

The focus of the presentation and description of the conversation (and later action) types below is not on students’ ability to make shifts among different kinds of conversations. Rather, it is on the context in which these conversations happened in addition to the descriptions of their characteristics. This is not to underestimate any possible implication for students' abilities to "sense" what kinds of conversations or actions are more appropriate for a situation and start using them or what makes particular conversations and actions "more appropriate" for students. On the contrary, the focus in this chapter is on students' abilities to use particular types of conversations and actions consistently in particular contexts. This presentation and discussion is perhaps as important as the discussion about abilities to make shifts in their work with CPEs, because it highlights possible different ways that students can or tend to use CPEs in the context of modeling natural phenomena.

3.2.1. Conversation type I: Talking about their program's structure

Prior to any work with computers, the students and I had conversations about their program ideas. In some cases the whole class had the same situation to think about and in other cases different groups of students were presenting their own situations and what programs they were thinking about writing. In these cases, the conversation was in a whole class format and all students were participating.

During conversation type I, I have identified several characteristics that were common among all groups and in all of our discussions of this type. i) Students used the program language as a communication medium, ii) students avoided talking about details of their programs, iii) students talked about ways of representing the main ideas in their program and iv) students described the simulation to support their program decisions.

Use of program language as the communication medium

In these early discussions, students saw their role as building computer programs rather than as building models. They talked about their program plans using the program language as a communication medium. They used known primitives such as forward, backwards right, left repeat etc, to describe how their program would be (see episode MW1).

Avoid talking about details of their programs

The students did not talk, however, about details such as the amount of forward or the degrees of a turning angle. Even when some students started talking about such details, someone would indicate that this was not the purpose of the discussion. For example, Samir chastised Aaron for providing the amounts forward and waiting:

124. **Richard**: the program was talkto, to jump2 talkto t2…
125. **Richard**: yea, yea, yea.
126. **Aaron**: fd 0.5 wait 1, fd 1 wait 0.5.
127. **Samir:** you were supposed to make a picture not write the program!
128. **Aaron**: wait.
129. **Richard**: but wait, Aaron, look, look, see what I, wait Aaron I want to show you guys what I did in mine. I did, less waits and the same fd, but less waits, to have, to look like they're <inaudible> gravity. So, when it jumps on moon he'll go <gestures showing someone jumping> instead of <gesture showing someone jumping quicker>

(excerpt taken from Aaron & Richard whole class presentation, 13 November 2002)

Talk about ways of representing the main ideas in their program

As briefly shown in the above excerpt (lines 124-129), a third characteristic of the conversation type I is that students talked about ways of representing the main ideas in their programs. Aaron and Richard's group, for example, talked about their program of having a boy jump on the earth and an astronaut jump on the moon. They talked about how they decided to keep the same amount of forward throughout their program but decrease the amount of waits, which was a representation of what jumping (in both the earth and the moon) is like. They did not talk about the decrease rate or pattern of waits nor did they talk about why that change occurred. Interestingly, Aaron and Richard did not talk about the differences between the program for jumping on the earth and the program for jumping on the moon. They preferred to talk about the similarities of their programs, which were similarities in their programs' structures: both programs were representations of "jumping motions", which have the particular characteristic of slowing down, as the distance from the ground becomes bigger.

Similarly, in presenting their ideas for representing an arrow traveling in the air, Joe and Samir talked about having 3 subprograms, each one representing a different part of the arrow's motion. They also talked about why it would make sense to write a number of smaller programs instead of a single one to represent a shooting arrow. Joe and Samir also talked about the structure of their programs, without providing any details about the specifics of their subprograms (see episode MW1).

Talk about how the simulation would look to support program decisions

Students also talked about how the simulation they wanted to develop would look. Talking about the simulation had a different focus from talking about their program, although it was meant to support their program ideas: students talked about how specific program ideas (mostly regarding program structure) would result in a particular simulation, mostly providing a description of their simulation. Students described their simulation in order to support their program decisions. For example in line 129 above, Richard talked about the structure of their program, and then supported it by talking about the simulation it would result. The way their simulation would look was a result of the particular code that created it. Therefore, if the simulation looked "realistic", then their program would be appropriate for representing the phenomenon. Similarly, Joe and Samir indicated that three different (sub)programs would result in a simulation where the motion of the arrow would have 3 distinct “parts”: one going upwards, one traveling horizontally and another one traveling downwards.

Below I present an episode from the whole class conversation that we had at the beginning of phase II of the study, focusing on Joe and Samir’s group presentation. I present some transcript and some analysis of student conversations to illustrate the aforementioned characteristics of the conversation type I.

Episode MW1, Discussion before programming, 13 November 2002

Episode MW1 took place during the second meeting of phase II of the study. During the previous meeting students played around with various programs available on the MW website and brainstormed about possible ideas they would like to program. I told students that the only requirement was to build a program that would represent a phenomenon from science, to show to their classmates and other people.

The discussion below was about Joe and Samir’s program ideas and it started while we waited for all the students to arrive in the computer lab. Joe and Samir arrived early and we started talking about what they were about to start working on. Like all the others, they had previously talked about several possible ideas and I wanted them to start thinking about a single one.

130. **Samir:** What if we show a throwing knife, or an arrow going through the air.
131. **Loucas:** an arrow going throw the air?
132. **Samir:** to a target, like there is a target and should the <inaudible>
133. **Aaron:** yea it’s like a target practice game um, a <inaudible> game like …. Your archer…..
134. **Samir:** wait, no, no, this is me and Joe’s idea. Me and Joe we must <inaudible> Ok, they have like, you have like a <inaudible>
135. **Aaron:** cut through air….
136. **Samir:** <inaudible>and threw <inaudible>and it powers <inaudible>first time this thing goes like this and chose direction in aiming your bow.
137. **Loucas:** ok.

138. **Samir:** and then when you click first click is stops there. I know how you can do it. And then I will have <inaudible>we have like a bar <interruption from speakers> or we can do it.. we can <inaudible> stop <inaudible>going back and harder is gonna shoot, I mean straight <inaudible>
139. **Joe:** me and Samir.
140. **Loucas:** now that, to program that is quite difficult.
141. **Samir:** so that could we just program an arrow from the internet?
142. **Loucas:** right, so what would be, what would that show, just, I mean, …
143. **Samir:** just this, the idea of if you shot an arrow it wouldn't go, um, and then it's, um, cutting through the air.
144. **Loucas:** ok. And Samir how would you program it though? It's, it's not fd, it's not, how, are you, do you have an idea, if you have an idea….
145. **Joe:** at the beginning it has to do right something, so that it….
146. **Samir:** so that the arrow doesn't go like this.
147. **Loucas:** so do you want….
148. **Joe:** at the very beginning, at the very first fd <inaudible> right something so that <inaudible>

Joe and Samir seemed ready to start thinking about the specifics of their programs. The conversation thus far was full of ideas about parts of a possible program, expressed specifically in code, using the various tools that are available in MW (e.g., sliders – what Samir referred to as bars). Some of the conversation was about how the simulation would look, but their focus was on how the motion of the arrow would be as a result of their proposed program.

After providing some time for individual group work, I asked each group to present their ideas to the rest of the class. The conversation below resumes when Joe and Samir started talking about their program.

175. **Samir:** ok, what we're gonna have is an archer who's, you could do this later <gestures showing changing the angle of the shooting arrow>, first we just want to make it so that when you click the archer shoots the arrow and we're gonna make, we can make one big program and all of <inaudible> stop it, or we can make a bunch of little programs to make it, cause that wouldn't be as realistic or we, I had an idea of making….
176. **Loucas:** wait, why wouldn't a bunch of small programs be realistic?
177. **Samir:** because when you have to do…
178. **Joe:** because the arrow would go hum, hum, hum. <gestures showing an upward motion of the arrow, a horizontal motion, and a downward motion>
179. **Samir:** … you can't make it turn. So it'll be like this dat, dat, dat.
180. **Richard:** no, if you had no waits it wouldn't be!
181. **Samir:** yea, but it wouldn't be…
182. **Richard:** or you can have small waits and it'll just look, <gestures>…. And not go …<gestures>. You can have .0001.

Joe and Samir started their presentation by talking about the structure of their program. They talked about clicking the archer to shoot and indicated that they thought about changing the angle during flight – but they would talk about it later. They also talked about the structure of their programs, indicating that they could either write a large program that would take care of the whole trajectory of the arrow or several smaller programs, each one during a different part of the arrow's trajectory (line 178): one during the upward motion, one during the horizontal motion and one during the downward motion (see figure 3.1).

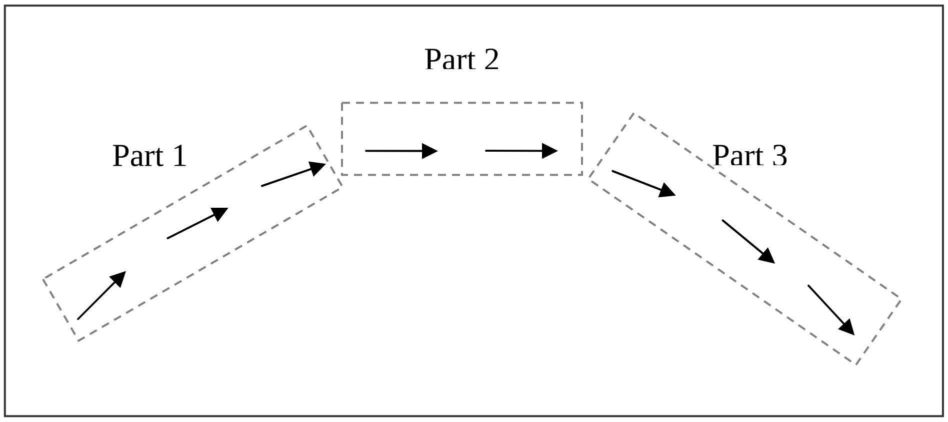


Figure 3.1. Joe & Samir's program structure

When Richard jointed the discussion, he focused on how the simulation would look (adding small waits so that one can actually observe the different parts of the programs – lines 180 & 182), whereas Joe and Samir were talking about the structure of the code, based on what happens in real life or how they wanted their simulation to look. In line 175 Samir argued that writing a single program "…wouldn't be realistic" and Joe justified Samir's argument by comparing the output of the simulation of one program to the simulation of several small programs.

Joe and Samir started to represent different parts of the motion of the arrow in their program, as parts of their program. In this sense, they started thinking about ways to program an arrow moving in the air by breaking the phenomenon into parts that shared common characteristics about the arrow's trajectory. Of course, one can argue that those differences are not related with the actual mechanism that underlies the phenomenon. Nor do they reflect any changes in the science that they represent; they only represent changes in the direction of the arrow, which are changes of how the physical phenomenon looks.

186. **Samir:** …we had this really good idea that we want to share with the two programs. What we can do is that we're thinking that we can make a little <inaudible> you know how this <inaudible>, we can make something to adjust instead of the wait, or the fd, we can adjust the angle, so that makes more and more and more and more, and then we can stop it when it's about to go down, and then make a program, minus-ing it…
187. **Richard:** bk
188. **Samir:** … so it goes down, down, down, down.
189. **Richard:** or bk. The, um, run the program backwards.
190. **Loucas:** (to Richard) no, but they are talking about the angle.
191. **Samir:** no, but then the arrow would be like this Richard.
192. **Loucas:** you're talking about the angle of the, of the arrow.
193. **Samir:** yea, cause if there's something that change the angle so that it gets <gesture> and then it'll go…
194. **Loucas:** do you agree, do you agree Joe? Do you agree with that?
195. **Joe:** yea, that, that …
196. **Samir:** and then also Joe has this idea of making an oval and then stopping it like right there, so that it would be half an oval and then it'll be like …
197. **Joe:** see, since we can make an oval, we can probably make half an oval. And…
198. **Aaron:** yea, and then half an oval would be the curve of the arrow.
199. **Loucas:** oh, you're, ok. By oval you mean that, the, the, the um, the direction of the, of the arrow, right?
200. **Joe:** yea, I mean, I mean like the arrow makes it that shape and then it goes from that point to target.
201. **Loucas:** right, ok. Ok. Good.
202. **Joe:** and then I was also thinking, if we had extra time we could try and make it so that the program throwing the arrow hits the um target, it might be able to make half of the arrow disappears so it's like it was stuck in the target.
203. **Loucas:** ok.
204. **Joe:** to make it more realistic.
205. **Aaron:** and also part of the arrow to disappear, maybe cause just the <inaudible>
206. **Loucas:** um, Joe, we are gonna have plenty of time because we are not finishing them today. We're gonna have next time and the time after that. So don't worry about having time. Ok?

As their presentation continued, Samir got into more details about their program, talking about writing code for changing the direction of the motion of the arrow (angle).

He avoided, however, to talk about details of their code (how much the angle would change in each program, or how much forward would the arrow move before slightly changing angle).

As the conversation continued though, Samir talked about Joe's idea of making a program that would "do" half an oval, to resemble the trajectory of the arrow (see figure 3.2). They probably took this idea from a previous session (three weeks earlier), when students discovered how they could write code that makes circles and ovals. They adjusted that idea to their program needs, indicating that they only needed half of the oval. This idea contradicted their initial idea about several small programs, because a single program can create an oval-shaped trajectory. However, this idea was another representation of a possible program's structure. Unlike their first idea, in presenting the oval trajectory the students talked about a single program that can create the motion of the arrow. The essence of a program that creates an oval shape is a mechanism of changing the turtle's (arrow's) direction. In the case of the arrow, that particular mechanism can be a representation of the mechanism that is causing the change of the direction of the arrow moving in the air. While focused on creating a simulation that would look like having an oval-shaped trajectory, the students were including, most probably unconsciously, a representation of the mechanism of the change in the arrow's direction.

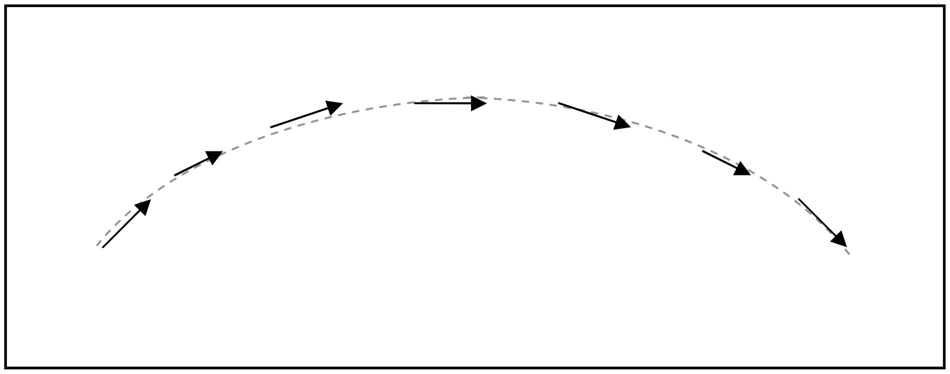


Figure 3.2. Joe's idea for the program

Towards the end of this conversation (line 104), Joe indicated that if time permits he would like to write code that would make the arrow's front disappear when it hit the target, in order to look more realistic. Like he said, he was thinking to pursue this concern only if there was enough time, possibly indicating that it was not a major issue. As he noted in line 106, this was to make their program more realistic, possibly indicating that the depiction was one of the factors that concerned them.

3.2.2. Conversation type II: Talking about program details

The second type of conversation occurred in small groups, while students were sitting in front of the computers. It was a conversation that I usually prompted, with a similar question that I use to prompt the previous type of conversation: "what are you thinking of programming?" Students however, responded differently to my question, and in all cases our conversation was more like a preparation for programming. Instead of talking about the structure of their programs, students talked about writing code to represent particular programming ideas.

The second conversation type was a "logistical" but still a "technical" conversation. It was "logistical" because students were focused on transferring their ideas into specific code. It was technical, because students' communication was based on the exchange of ideas expressed in the program language. During this conversation, students started to talk about some the details of their program, even though once again they seemed reluctant to talk about their program in that kind of detail.

That students were in front of the computers seemed to be an important change of context, because students were not simply planning their programs: they were about to start typing. During conversation type II, they were talking about different ways of programming particular programming ideas. Type II conversation i) was usually a prompted conversation, ii) was about how to write a program, and iii) included references to how the simulation would look.

Conversation type II is a prompted conversation

Conversation type II occurred only when I prompted a discussion about students' work. As I discuss in the conversation type III (talking while programming), students usually deferred discussions about their programs and program details after they had some code written down. I had a conversation about a future program with all individual groups at some point, but in all cases I prompted the conversation. In fact, during analysis of student conversations and contextual inquiry, I could not find any conversation in a small group about a program to be written; all such conversations were either after a program was typed or prompted by me. Even within the context of type II conversations, there were instances were students thought of an idea, and started typing instead of sharing first, as the following excerpt shows:

58. **Samir:** how do we make it go fd that many times and then done that?
59. **Loucas:** I'll show you.

60. **Joe:** you just keep going fd, but you change the angle right here, so change the angle to be like 1 point….
61. **Samir:** oh, I have an idea! Ok, repeat … <Samir started typing>
62. **Loucas:** wait, wait, tell us! Tell Joe!

(excerpt taken from Joe & Samir group conversation, 13 November 2002)

Given the fact that I usually prompted this kind of conversation, I do not claim that students could initiate this of conversation by themselves or that they were unable to have such a conversation. Rather, I argue that students were capable of such a conversation, in which they talk about their particular program ideas, including ideas about the structure of their programs but mostly about the details of the code in their programs. In both conversation type I and II students were talking about their program plans. However, in this second type, students seemed to be a step closer to the process of typing up their program.

How to program a specific idea

Students' concern during type II conversation was how to program a specific idea, which resulted in students talking mostly about what code to write. Students were mostly using the program language to communicate ideas, even though as figure 3.3 shows, there were some conversations about the code itself and some without any use of the program language. Students were now more specific about the details of their code and as shown in figure 3.3, they both talked while using the code and talked about the code.

In one way, students that previously (conversation type I) were concerned about the structure of their program were now concerned about the details of the code in each of their programs. What had possibly caused this difference? It is possible to suggest that since students were now sitting in front of the computers, they probably had started thinking about specific ways that they could write their programs.

Using the simulation to support their program ideas

All these conversations were also accompanied by some discussion about how they wanted their simulations to look, in an effort to justify all their program decisions. Students talked, for instance, about how they expected the arrow to travel in the air or how they expected jumping on the moon or on the earth to look, in order to justify particular and detailed decisions they were making about their programs.

There is an important distinction between using the code to justify a proposed program (e.g., this is what is going to happen as a result of these particular lines of code) and describing the scenario and trying to translate it into code (e.g., we will have balloons flying up, so we would have a rule (code) that would make the balloons move upwards). Line 63 from Joe and Samir conversation is a representation of this type of conversation:

63. **Samir:** ok, repeat, repeat 5 time with the angle going up by let's say 10, so that will be 60, so then it'll be 55, then it'll be 65, so we're going like this <gestures> and it'll stop here and then make another program that says repeat 5 but it's going down by 10, so it'll be like this <gestures>

Samir presented details of what he thought of typing, talking about what that program would do by referring to how it would look on the screen (describing the "scenario" of the simulation) and then continued talking about the rest of the program. He also indicated that he was talking about two programs, utilizing their previous ideas about the structure of their program. Below I present the episode represented in figure 3.3, as an example of type II conversation (Episode MW2).

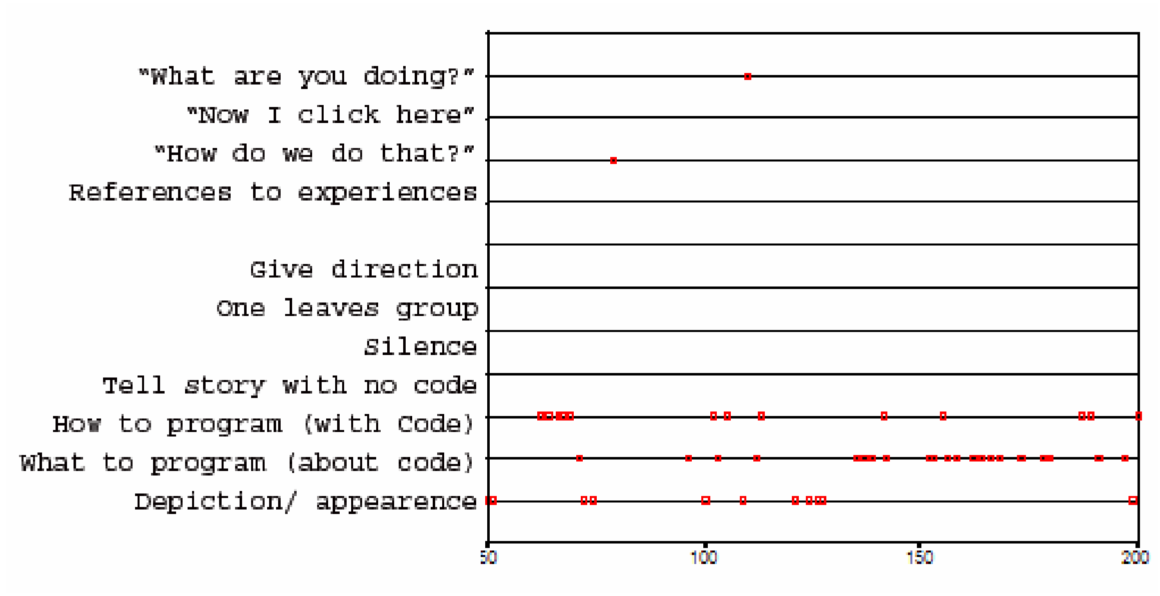


Figure 3.3. Conversation type II
(Source Samir & Joe group, 13 November 2002)

Episode MW2, Preparing to write a program, 13 November 2002, Joe & Samir

Episode 2 presents a conversation that I facilitated in Joe and Samir's group. The conversation started when I joined the group. Joe and Samir had just started typing some code for their program. I asked them about what they were planning to type.

18. **Samir:** um, what we were thinking goes like this…<gestures showing the motion of the arrow>
19. **Joe:** we were thinking, ok, making a half oval shape.
20. **Loucas:** ok
21. **Joe:** go like that <gestures>
22. **Loucas:** so, so how would you start?
23. **Joe:** well, we would probably like we probably <inaudible> at the beginning right, in front of that, so that the angle changes, because if we start with the fd and then fd and then if you do the right, right something like <inaudible>it would go like this.
24. **Samir:** well, is there, I have a question, is there a way to say, make is so that the angle goes higher and higher and higher, and then stop and then done with that?

Once again, from the beginning of our conversation, Joe and Samir were specific about the particular programs they were thinking to write. They presented their ideas as if they were filling in blanks in their programs: they already had talked about the structure of their programs, specifically talked about different programs for the different directions that the arrow would move. Now they were talking about how to write those separate programs. They even asked me about a way they could have the program change the angle that the arrow travels, possibly indicating that they started to think about the motion of the arrow in terms of step-by-step changes in the arrow's position and angle.

Despite of the fact that students were now talking about particular code to use in parts of their programs, in some cases they seemed to prefer typing that code instead of talking about it.

54. **Samir:** repeat like 5 times, cause it'll, we want it to stop half way. Or do we want it to all the way? 'Cause we can't really make it go down and up.
55. **Loucas:** what do you think Joe, do you want to stop half way, or do you want to go all the way?
56. **Samir:** if we go all the way it'll be like this <gestures> and <inaudible>again because we can't…
57. **Loucas:** why not?
58. **Samir:** how do we make it go fd that many times and then done that?
59. **Loucas:** I'll show you.
60. **Joe:** you just keep going fd, but you change the angel right here, so change the angle to be like 1 point….
61. **Samir:** oh, I have an idea! Ok, repeat … <Samir started typing>
62. **Loucas:** wait, wait, tell us! Tell Joe!

This was a familiar mode of work. Students working with MW preferred to talk about already typed programs instead of talking about programs that they were thinking of programming. As I discuss later, choosing to talk about a written program had the advantage of having something written down that they could talk about (possibly making it easier to have a conversation about something specific), but at the same time the disadvantage of making the action of typing a "lonely process": one student would type and the other member(s) of the group would just watch or even leave the group and return later.

The conversation continued in another familiar way:

63. **Samir:** ok, repeat, repeat 5 time with the angle going up by let's say 10, so that will be 60, so then it'll be 55, then it'll be 65, so we're going like this <gestures> and it'll stop here and then make another program that says repeat 5 but it's going down by 10, so it'll be like this >

64. **Joe:** the only thing is that is coming here <gestures showing to the horizontal portion of the motion> then it kind of <inaudible> <gestures>
65. **Samir:** well, let's just try it so…
66. **Joe:** ok.

Joe and Samir had a disagreement about a particular piece of their program that Samir was proposing; or at least they were not sure which of the two possible ways to use. Samir suggested writing two programs, one for the upward motion and another for the downward motion. The first program would stop as soon as the arrow becomes horizontal and the second program would start at that point. Joe was concerned about what would happen when the arrow reaches the point that becomes horizontal. Earlier, he suggested a third program for the horizontal part of the arrow's motion (line 178 in Episode MW1).

Instead of further debating this, Joe and Samir thought that they could simply try their idea and then decide whether one was better than the other. This was a common reaction: students in the study preferred writing and testing code to see how it looks rather than reading and talking their way through the code.

As the conversation proceeded, Samir talked about writing one program until the arrow reaches the top of its projectile and then another program for its downward motion. Joe, on the other hand, started thinking about having two programs while the arrow was moving upwards, one while the arrow would be moving horizontally and another two programs coming down, showing with his hand that there were 5 different directions in the arrow's motion.

Thus far, the conversation was about two different things: about the structure of their program and about some details of the code of their programs. In talking about the structure of their programs, students talked about specific numbers of programs (rather sub-programs or sub-routines) to represent particular parts of the arrow's motion. However, the parts of the phenomenon that they were talking about were simply "visual parts" of the phenomenon: they were identified by the changes in the visual characteristics of the direction of the arrow's motion.

In talking about the details of their programs, students had barely started talking about the particulars of each of their proposed subroutines, possibly helping them to think about the science of the phenomenon and what it was causing it. Of course, in several occasions they talked about specific number of repeats, or amounts of forward (e.g., line 63), but they seemed to use those numbers as examples of ideas: whether it was going to be forward 5 or forward 8 it did not seem to be important. In fact, as soon as students started talking about the particular code of their programs, I surprisingly found out that students were not clear about the number of repeats, the amount of forward or how much the direction of the angle would change in their programs.

82. **Loucas:** so, I'm gonna put something there and we can, how much [number of repeat times] do you want to put for a start? 10? 5?
83. **Joe:** yea, 10.
84. **Loucas:** so, <inaudible> what we should write?
85. **Samir:** um , fd, I don't know you decide the numbers.
86. **Joe:** fd goes like about um…
87. **Loucas:** I just put it randomly.
88. **Joe:** oh, ok, fd 15, well then it's going to stop every 15.
89. **Loucas:** ok.
90. **Joe:** it should be a large fd, like…
91. **Loucas:** a larger one? Why?
92. **Joe:** like <inaudible> so that it'll go like this <gestures> because if, if it just go straight <gestures showing more fd but smaller distance>, well, that would, yea, that would work.

Samir did not seem to care about the numbers, referring the decision to Joe (line 85). Joe was not clear either. A larger number of smaller forwards seemed to make more sense to him at the time, but he was ok with the larger number of forwards that I suggested earlier. Earlier (in episode MW1), the same students were arguing about the structure of their programs, documenting their decisions using observations from everyday experiences of how things look in real life. I was wondering whether they could have a similar conversation about the code of their programs.

At the time I thought that the above was a matter of perspective. Joe and Samir were in a “state of mind” focusing on their program’s structure and not on the details of the code. Minutes ago, Aaron and Richard were having a conversation about their code and I asked them to join Joe and Samir’s group to help in a discussion about the arrow’s program.

99. **Loucas:** Joe and Samir want to write a program that an arrow would go like that.
100. **Richard:** yours so easy!
101. **Loucas:** tell us!
102. **Richard:** ok, you guys know how we made the circle! You go fd 1 then you go wait, or um what was it? Fd 1 right 1 fd 1 right 1 fd so on and so on, so when you got here you go, you keep on going, you keep on going.
103. **Joe:** so right here you go repeat 100…
104. **Loucas:** so Richard, if I want
105. **Joe:** and then it’ll go like this.
106. **Richard:** there’s no waits!
107. **Loucas:** Richard, if I want to make a, to circle, how what should I put?
108. **Richard:** fd, right, repeat, repeat 1, repeat 100, 100, 100, um [fd 1 right 1]. We did that to make a circle! Look, what it does!
109. **Loucas:** so, this makes a circle, Now they need to make this kind of shape, though, it’s not a circle, what do you call this shape, oval?
110. **Many**: oval.
111. ?: football!
112. **Richard:** when you go fd .5 and then right .5,
113. **Loucas:** to oval
114. **Richard:** or you go fd 1, um repeat 100 fd .5
115. **Loucas:** why less?
116. **Richard:** ok, fd 1!
117. **Loucas:** why, why? So, come on, because I am not sure what Richard is saying.
118. **Richard:** fd 1, ok, fd 1, then right um .5.
119. **Loucas:** .5?
120. **Richard:** yes.
121. ?: and then 200 won’t do it.
122. **Loucas:** ok. I say, I say, please don't do that, do you agree with him? What, what this would make? What would this program make?
123. **Richard:** remember of, what is me and you or was it just me by myself that made the weird <inaudible>
124. **Aaron:** it was you and Joe
125. **Joe:** it was me and you!

126. **Richard:** remember when we made the <inaudible>
127. **Joe:** yea.
128. **Richard:** we did this, to make a circle.
129. **Joe:** no, no, that weird oval shape that was the circle shape.
130. **Richard:** oh yea, I know, but we had the .5, Aaron, Joe, Joe, we, I put .5 and it said it won! It will go less every turn, you could do that.
131. **Loucas:** let me ask you this.
132. **Joe:** well, yea, maybe.
133. **Samir:** can we just try to see…

This last part of the conversation was different from the previous one. Right away Richard and Aaron started talking about how to write a program that would give them the particular trajectory that Joe and Samir were thinking. Richard and Aaron were also talking in detail about the specifics in the program: number of repeats, amount of forward, amount of turning angles etc.

While reviewing the conversation, however, I asked my self: is this conversation really different from the previous one? The goal of the conversation was to write a program that would result a half oval shape, the trajectory of the arrow moving in the air. Students were not concerned about the structure of the program any more; they just wanted a program that would result something that would look ok, the way they thought the arrow would move on the air.

3.2.3. Conversation type III: Talking while programming

For a large amount of their time, students in the study were typing and testing code, to write programs that would represent natural phenomena. Typing code was an important part of student work for this study, because it provided information about the actual use of the CPEs' tools by students while modeling. Conversation type III describes students' conversation during programming. During programming: i) the conversations were limited, ii) students talked about code and primitives, iii) students did not talk about the phenomenon under study, or how their simulation would/should look.

Limited conversation while programming

While typing code of a new program, students' conversations were very limited. As figure 3.4 shows, most of the time, only one student was at the group (indicated by category "one leaves the group"), typing code, whereas the other(s) (usually the groups consisted of 2 students each) would leave the group for a short time and then returned. There are at least two possible reasons that this was happening. First, students did not share and discuss particular program ideas in such detail that they could follow their fellow student who was responsible for typing the code. Because of that, the student that was not typing did not have anything to do during that time, and leaving the group might had been a more exciting thing to do that just sitting and looking at the code the other student was typing. This might be possibly supported by the fact that when both students were at the group, the student who was typing the code would talk out loud while typing, probably giving some idea of what he was thinking and doing at the time (this was observed in all three groups). Even though this might be an unusual phenomenon to observe, it seemed to help the other student follow what the first was typing.

A second possible reason would be that the process of typing code requires only one person at the keyboard. The process is so straight forward that the only thing that students need to do is typing up code. In fact, it is possible that talking during the time might be a distraction: code needs to be precise without any typos.

The student who was typing seemed to undertake the role of “a typist,” with at least one group (Aaron and Richard’s) explicitly specifying who the typist was each time. This could possibly add more pressure on the other group member not to interfere with the typist’s work.

Conversation while programming was about (program) primitives

During the act of typing code, students talked only to provide each other with help with the language primitives, correct typos and give directions as to where and what to type, in those cases that the student who was typing did not know. There was no conversation about what the simulation was, or about the particular details of the program. In this sense, their conversation was strictly technical, and only when some kind of help was needed. In all the other cases, students let the “typist” type their program.

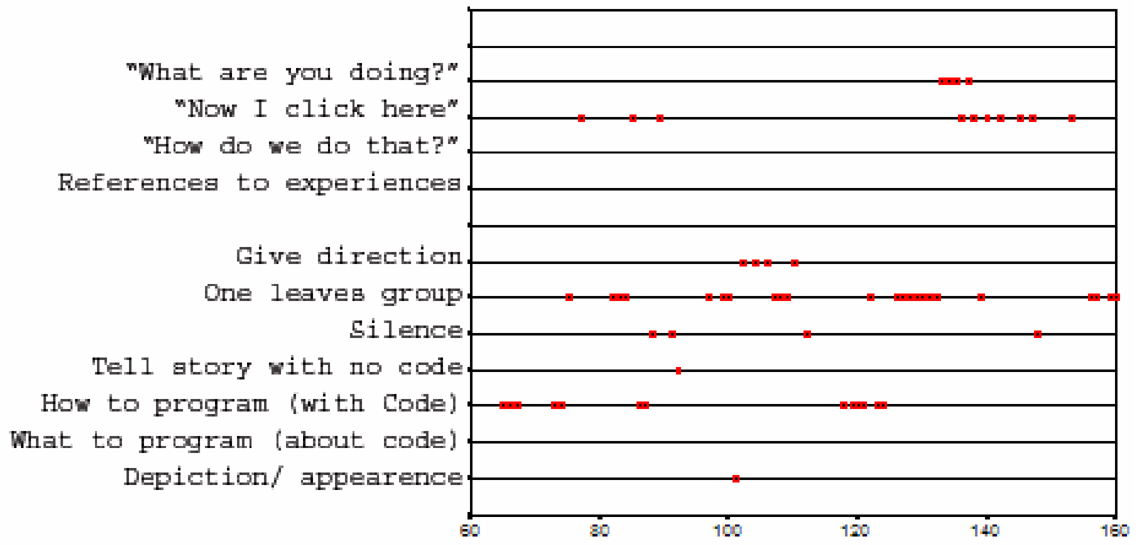


Figure 3.4. Conversation type III
(Source Aaron & Richard group, 13 November 2002)

Before presenting an episode of conversation type III, I would like to highlight one of the points that I mentioned earlier. During programming, students were not talking about their programs; rather their conversations were about writing up a program with no typos. However, as seen in previous types of conversations, students could have conversations about their programs by describing program ideas, describing their program's structure and supporting program decisions by referring to the simulation and to experiences they had from everyday life. Thus, it is possible to suggest that students had abilities to have different types of conversations based on the need of their current actions. Typing code is the process of typing correct primitives, and students seemed to let one handle the task. As I discuss later, after typing a program, students talked about details of their programs, what the simulation looked like and what changes to make in their program to represent more accurately the phenomenon.

Episode MW3, Conversation while programming, 13 November 2002, Joe & Samir

Episode MW3 is a continuation of episode MW2. In this episode Joe, Samir and Richard were typing code in MW. While typing code, students became less vocal than before, leaving most of the work to the "typist." The transcript below starts while Samir was typing.

161.**Richard:** right 0.5 End of quote.
162.**Samir:** now watch Richard.
163.<silence while typing>
164.**Samir:** Shoot! <testing their program>
165.<silence while looking at their code>
166.**Richard:** how do you talk to, are you talking to a?
167.**Joe:** yea, we named it a, as opposed to t1 or t3
168.<silence while looking at their code>
169.**Richard:** turtle!
170.<silence while running the simulation>
171.**Richard:** I don't know how to a in shoot.
172.**Samir:** quiet!
173.<silence while looking/changing their code>
174.**Samir:** yea!
175.**Richard:** do you want it to turn?
176.**Samir:** we want it to go like this. Ou!
177.**Richard:** it's forward, I can do that. I can do that! Yea!
178.**Samir:** o ho! Richard can do it, Richard can do it! Richard's doing it! Richard's doing it!

179.<silence while typing>
180.**Samir** : <inaudible>
181.**Richard:** weeeeee! And may the <inaudible> (Richard leaves the group)
182.<silence while clicking/typing/looking on the screen>

During programming, students avoided interaction. When there was some interaction, students were focused on writing a program that would run. This was probably the reason that students mostly had technical conversations about typing something that would have been more appropriate than typing something else. For instance in line 166 where Richard questioned Joe and Samir’s program using a name for a turtle instead of the default t1. This had nothing to do with the phenomenon represented; rather, it was related with the logistics of getting a program to run. During these conversations, students were not concerned about the phenomenon they were trying to represent. Neither their program’s structure nor how the simulation looked was part of their conversations.

In addition to their limited conversations, students were using “trial and error” techniques: they typed a program, tried it out, then went back to the program window to see what was causing the bugs, made changes and then went back and tried it and so on, until the program successfully ran. Running their program was a quick way of finding out whether their program was running successfully.

During this part of their work, Samir was mostly the “typist” and Joe had limited involvement in the group. This was another familiar theme in students’ work with MW. Students, as in this case, tended not to share the details they were thinking of programming (Samir told Richard to be quiet – line 172). Instead, they tended to work alone, type the program and then talk about it. This was also supported by Joe’s actions during typing code. He did not participate in this process. He would sit for a few seconds and then leave the group, to come back later and see what Samir was doing. Even though students started typing code around line 159 (with some of the utterances being a few minutes long while students were simply not talking) only in line 207 Joe, while looking at the code, indicated: “Oh, I get it, we are making a circle shape than an oval shape. Plus the oval shape is going like that <gesture> not like that <gesture showing on the computer screen>”. At that point, Joe could read the code that Samir wrote and could understand what kind of program Samir was writing.

3.2.4. Conversation type IV: Talking about how the simulation looks

As soon as students had a program that was successfully running, they started having conversations about their program. This is the fourth type of conversation, during which students talked about a program they had already written, whether it represented the particular phenomenon in an accurate way. During this conversation, students i) talked about how the simulation looked, and ii) made references to everyday experiences.

Focus on how the simulation looked

During this type of conversation, students were not concerned about the actual code of the software, possibly because this type of conversation was carried out after they successfully wrote and debugged the code for their programs. Rather, their concern was whether the simulation was representing the phenomenon accurately. Their focus was on how the simulation looked, and most of the conversation, as figure 3.5 shows, was coded as talking about the simulation without the use of the program language. However, the focus was not on the actual appearance of the objects, background etc of the simulation, but on describing how different parts of the simulation looked or should look.

This new focus was probably a result of their work during the debugging of their program as I discuss later (see program strategies II: correcting depiction). After students successfully created a program that would run on the computer screen, they shifted their focus on making their simulation look "better," to look much more like the phenomenon. During debugging their focus was on the code; now that they had a program that ran, their focus was on the depiction.

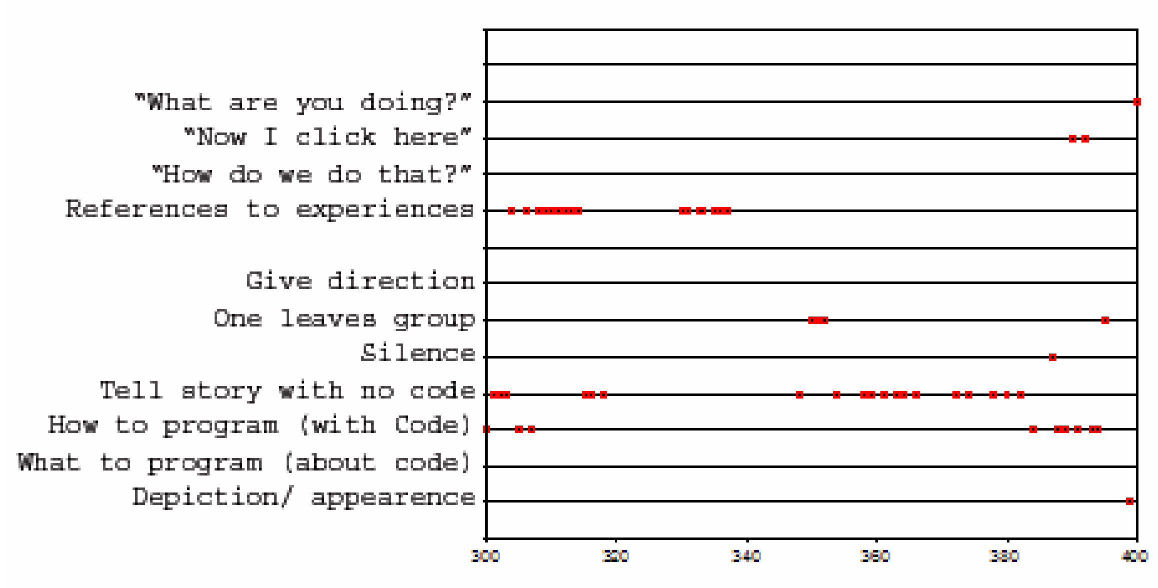


Figure 3.5. Conversation type IV
(Source Aaron & Richard group, 13 November 2002)

References to everyday experiences

During this type of conversation students used experiences from everyday life to debate whether or not their simulation was showing what really happens in the physical world. It seems that their goal during this phase of their work was to make sure that their code created a simulation that correctly depicts the phenomenon they chose. Because of this, their focus was on comparing their simulation with experiences they had had, in an effort to decide about possible changes to their program.

Episode MW4, Conversation about how their program looks, 13 November 2002, Aaron & Richard

This short episode of student conversation, starts just after Richard and Aaron had finished debugging their two programs: one for a person jumping on the Earth and another for an astronaut jumping on the Moon. They then asked me to join their group to see what they had. They ran their programs, and the differences of the two persons jumping were rather small, so I asked whether they could change them so that they would be more obvious. That, sparked a conversation about how their programs looked and should look, which is basically a conversation about whether their simulations looked ok.

224. **Richard:** Aaron, you need to change, I told you to change the waits to wait less in the others (on Earth), and you've made them longer, because their wait should be way less because he jumps on the moon.
225. **Aaron:** he's on earth, so it would be way more than wait.
226. **Richard:** so it'll going like zink! <gestures showing quick>
227. **Aaron:** the wait.
228. **Richard:** tink, tink <gestures showing quick>
229. **Aaron:** no, Richard, the moon has way less gravity.
230. **Richard:** I know, the wait, how long it waits. So it's like toush! <gestures showing quick>
231. **Aaron:** it has less gravity. So that the moon has…
232. **Richard:** the moon has more wait, I mean the moon has less wait that this thing [Earth] has. So this guy waits until he's coming slowly…
233. **Aaron:** stop talking in the computer language. Does the moon have way more gravity? Um less gravity?
234. **Richard:** yea.
235. **Aaron:** and the earth has way more gravity.
236. **Richard:** yea.
237. **Aaron:** so the moon is like 4 times smaller than the Earth. Ok?
238. **Richard:** ok
239. **Aaron:** so, it would be, so on earth …..
240. **Joe**: so Richard, <inaudible>
241. **Richard:** Joe, he [Aaron] is making the guy on earth going ouuuuu <gestures showing quickly>, and the guy on moon go ou. <gestures showing slowly>
242. **Aaron:** that's… it makes sense!
243. **Richard:** no he is making the guy on earth going like (slowly) and the guy on moon go like (fast).

The purpose of this conversation was whether their simulation showed accurately the phenomenon they wanted to represent. In this case Richard disagreed with Aaron's program, because his program was showing the boy on the Earth jumping much slower than the astronaut on the Moon. Richard was suggesting that their simulation should show the astronaut jumping slower than the boy on the Earth, showing with gestures how that motion should be. Aaron on the other hand was thinking that because of the more gravity on the Earth, their simulation should show the boy jumping slower (more waits in the earth program). To support their ideas, Aaron and Richard talked and compared their simulation with how the phenomenon in real life looked like. They were also referring to particular lines of code to suggest possible changes they thought and to real life experiences to support their ideas.

Usually, this kind of conversation was short, because students moved on to different kinds of conversations in order to resolve their differences. Had they decided to make the changes, they usually started a conversation while making changes in their program, as I discuss later in the program strategies III.

3.2.5. Conversation type V: Talking about what happens in the simulation.

A fifth type of conversation was identified when students were showing their simulation to others. This conversation was not observed on a regular basis; it was only observed when one student was absent from the group at a time and the rest were showing what they did, or when students were presenting to each other their final products. I saw two groups of students presenting their programs to others (Joe & Samir, Jiana & Gabriella) and for this reason, I do not present this type of conversation as an emerging type of student conversation, but rather as a possible conversation that can occur in this particular context.

During conversation type V students talked about what happens in their simulation, and made no references to the code that created the simulation.

Talk about what happens

This type of conversation was characterized mostly by talking about what is going on in the simulation, without much reference to the code that generated the simulation. Students talked about their simulation, and in particular what was showing, in a way that resembled telling the story of what happens in their simulation (similar to conversation type I in SC findings, see section 4.1.1). As figure 3.6 shows, there was only minor use of the program language.

This is interesting, because students' work and conversation with MW included a variety of foci, and during parts of their work were focused on the structure of their code as representation of the phenomenon (conversation type I, section 3.1.1), focused on the code that created a program that could successfully run (conversation type III, section 3.1.3 and program strategies I, section 3.2.2), focused on the simulation and its depicting details (conversation type IV, section 3.1.4) and focused on the code as a representation of the phenomenon under study (program strategies III, section 3.2.4).

In the two situations that students presented their designs to others, they had finished typing and debugging their code, dealt with depicting details and had some discussion about how their code could be/was representing the phenomenon under study.

Still, in presenting their programs/simulations students chose to talk about what was going on in their simulation and run their program to show others.

Code was used as a tool to create a simulation

A possible hypothesis that data from the study seem to support is that students can use tools that CPEs provide in a variety of ways. For instance, code was used by students to create a simulation, to communicate ideas or to represent a phenomenon. It is possible that in this case, students viewed the code only as a tool for creating the simulation, and not as a tool for further communication or for showing to each other. It was easier to show the simulation rather than reading and explaining the code.

In fact students' references to the code below were references to what particular programs or lines of code responded to in the simulation. As it can be seen in the following excerpt, Joe was talking about what different programs do (thus there is "some" reference to their programs) but mostly describe the simulation that the programs create.

175. **Joe:** well we made, um, a whole bunch of stuff, like, we made a program for a shoot1; shoot 2 is slightly different…
176. **Loucas:** tell him what shoot 1 is because he doesn't know what you talked about, and then we can…
177. **Joe:** shoot 1, shoot 1 makes the arrow to go straight forward because. It doesn't go up, but it moves down, like…
178. **Samir:** what's up with all the humps [refers to a stamp technique that I helped students add in their program that can make a dot on the screen for every position of the turtle in every tick of the program]?
179. **Joe:** the humps?
180. **Samir:** do, do, do, do
181. **Joe:** oh, they estimated this, what shape does it make.
182. […]
183. **Joe:** and then, um and then make it eventually, but we, at shoot 1 like it's, it's now, it is straight <showing the simulation> and you can't really see that it drops at all. Um, you can't really see that it kind of drops but um…

(excerpt taken from Joe & Samir group, 4 December 2002)

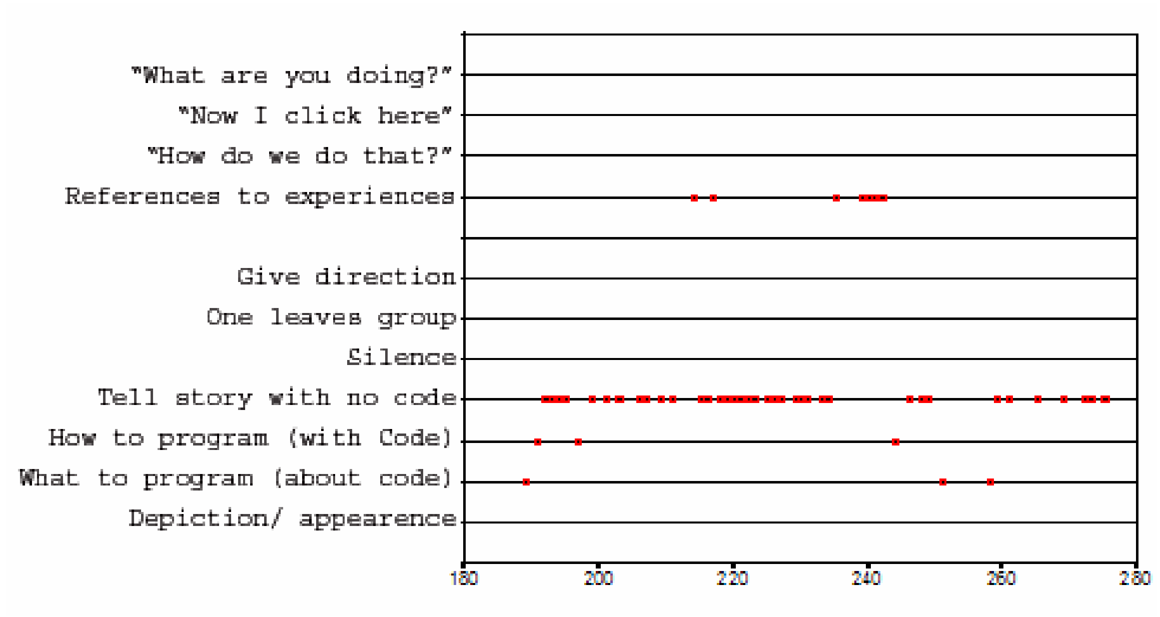


Figure 3.6. Conversation type V
(Source Joe & Samir, 4 December 2002)

That students seemed to use code as a tool for creating the simulation while presenting their programs/simulations, might have two implications for using CPEs in science for developing models of natural phenomena. MW is considered to be a traditional CPE that uses textual programming language as means for developing programs. As seen in previous types of conversations, there were cases in which students had very technical conversations about and with the program language (code). As soon as programming was done, students shifted their focus on the simulation (e.g., during the phase of correcting depiction and presenting their programs to the others). A first major implication is related with the iterative process of developing models, testing them and refining them. When students' focus after the first debugging shifts from the program to the simulation and depiction, the teacher might need to help students focus on the code again, in an effort to develop models that would show the mechanism that causes the phenomenon represented.

A second implication is that it not might be productive for modeling in science to have students present their programs to each other, especially if the only way students see this presentation is simply to show their simulation to the others. Findings from a pilot study (Louca, unpublished report) showed that when students presented to others working with a different CPE, they took them through their code talking about what each line represents. In that case, it is possible that it can be more helpful to have conversations about how different lines of code represent particular ideas. On the other hand, it is possible that the only thing that needs to happen is to provide students with extended experience with programming, which might make them more comfortable talking about their code in addition to talking about their simulation. However, this cannot be justified from results from this study and further investigation is necessary.

3.3. Types of activities

Students' work with computer was also analyzed for their activities while developing models of natural phenomena. Unlike conversational patterns, development of activity patterns was largely supported by watching students' work with the computers through their videotaped work and video-captured screens.

Three major types of activities were identified and are presented below. These include program strategies I: writing & debugging code, program strategies II: correcting depiction and program strategies III: correcting represented science. I also present a fourth activity type that was common among all analyzed groups, which, however, is not a program strategy: it includes students' activities when they started working on a new program, including dealing with the characters and backgrounds of their simulation.

The different program strategies that I identify and highlight below were possibly created by particular needs while programming. In different parts of their work (type a new program, debug their program, make corrections to depiction of their program or make changes in the code to represent the science of the phenomenon) students seemed to use particular programming strategies that were useful for that part of their work. For instance, to write a program and successfully run it is one task, and an important one, especially if you have no program or your program does not work. As soon as students successfully debugged their program, they could then turn to fixing the depiction of their simulation. This was a different task: students were not concerned about the code of their program any more, but they were attending to the depicting details of their simulation. The two processes (write and debug a program, and dealing with depiction) required different activity patterns and different focuses. It is not clear, however, whether the shift of focus occurred by the students who were ready to change mode of work, or by the different task.

In my effort to report the activity patterns, I present and describe them along with the context in which they occurred, provide details about student conversations and activities in the form of student conversations (transcript) and accompanied narrative. This is an effort to document different uses of the CPEs by the students and possible implications of such use for developing models/representations of natural phenomena.

3.3.1. Dealing with characters and backgrounds of their designs

Although programming in MW can include a range of activities, it can be mainly characterized as the process of typing instructions in code for characters (turtle(s)) to follow. The process of typing code is different from the process of running the simulation. The two processes require different actions: one is typing of one dimensional code and the other is running that code to create 2 or 3-dimensional graphical results. Students can also type code without having a character in place, even though you need one in order to run the code.

When students started working on a new microworld, they first spent some time designing or finding appropriate characters for their programs. Unlike students working with SC (see chapter 4), students working with MW did not spend a considerable amount of time setting up characters, and they returned later to deal with characters and backgrounds.

Short time dealing with characters

The time that students spent early in their work to find appropriate characters and backgrounds for their programs was relatively short: Joe and Samir spent about 15 minutes [4], looking for an archer on the web and then decided to draw one from scratch. Aaron and Richard spent about 7 minutes setting up their characters (see figure 3.7), since they found what they were looking in the character window of MW and Gabriella and Jiana spent 9 minutes looking for characters and then agreed to continue without any, until our next meeting (I promised to provide them with the characters they wanted).

During the time that students spent dealing with characters and backgrounds, they were focused on a single character, which they wanted to be perfect. Samir and Joe wanted to get an archer and they probably surfed on the Internet among 30-50 archers

[4] The time provided excludes the portions of their work that students were doing off-task activities. It only includes those times that they were engaged in the activity of finding and placing characters and backgrounds.

before deciding to draw one from scratch. As soon as they had something in place they moved on to type their code, even though there were some more characters to be included in other parts of their programs (a target for instance).

This is an important distinction from SC students' work. Students working with MW spent most of their time creating characters rather than looking around (in MW or on the web) in search for appropriate characters for their designs. Also, they could easily start writing some code, even though they did not have all the characters or the background that they wanted. Also, conversations about how their characters and their simulation's background looked were limited. Students working with MW did not seem to be concerned about how their characters looked. Even in the case that they were deciding to draw their own characters, their drawings were very limited in how realistic they were.

Students would return later to deal with characters

To finish setting up their characters and backgrounds, students returned in later parts of their work (during the same day or during the following meetings), while programming, usually after finishing a new program. Again, in those cases, students spent short periods of times (20-30 utterances, as the example in figure 3.8 shows) and returned to writing and debugging code. Before the break in timeline in figure 3.8, students were debugging their code.

Data from student activities during programming shows that students did not spend much time at the beginning of their work to deal with characters and backgrounds. More importantly, students did not seem to feel the need to have a complete set of characters for all aspects of their programs. Gabriella and Jiana had no problem continuing with writing some code without having any characters in place. Joe and Samir proceed to programming having a simple straight line with a pointing edge (representing their arrow), deferring for later drawing a better arrow and a target. While writing and testing their code, students usually returned to finish working with their characters, possibly when they needed to test their programs.

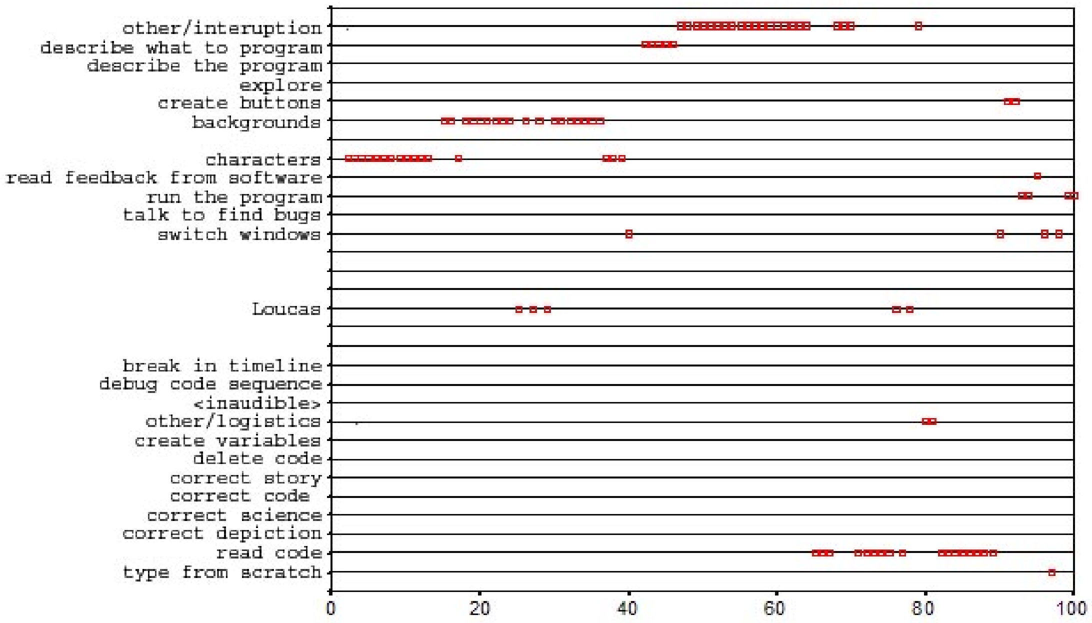


Figure 3.7. Student's actions in the beginning of their work
(Source Aaron & Richard group, 13 November 2002)

Another possibility that these findings may suggest is that students working with MW could break the process of developing representations of natural phenomena into small manageable pieces, even though those pieces were related to each other. For instance Aaron and Richard chose and added a character for jumping on the earth, went to write some code for that character to jump on the earth, and only when they finished debugging that code did they return to set up the character that would jump on the moon and then continued typing the program for the character jumping on the moon. They seemed to be comfortable finishing their program piece by piece.

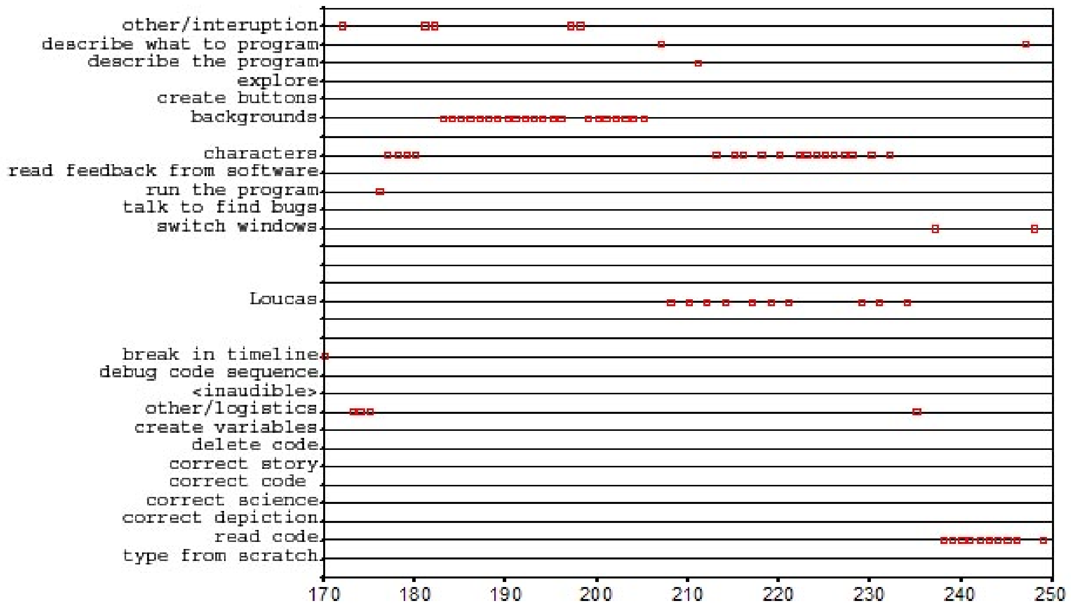


Figure 3.8. Students dealing with characters and backgrounds during the process of writing code (Source Aaron & Richard group, 13 November 2002)

3.3.2. Program strategies I: Writing & debugging new code

The first type of program strategies includes two distinct activities, writing and debugging code, which shared a common focus: writing a new program. Program strategies I, were usually observed when students started a brand new program which they typed from scratch. Generally speaking there were two major activities during this type of work: during the first one, students were typing code in the program window. During the second phase students were testing and debugging the code they wrote.

Writing and debugging code were rather “technical” activities and students’ focus was on the code itself. This means that students were only concerned about the code and their conversations were about using the right program primitives. This is important because during this part of their work students were simply trying to write a program and then successfully run it. Because of that they were not using the code as a tool for either creating a simulation or for representing a natural phenomenon.

Writing new code (Uninterrupted work & limited conversations)

As soon as students started writing a new program, they spent several minutes in the program window typing up code. During that time they did not interrupt typing for either testing the programs, or even for talking about the program they were typing. As I indicated earlier in “conversations while programming” (section 3.1.3) students did not talk much during that time: one was typing and the other was waiting for the “typist” to finish her work.

Three possible reasons might have caused this. Students were accustomed of writing a large program instead of breaking it into small sub-routines, even though when they talked about their program plans they talked about different parts of their program to represent parts of the phenomenon. Instead, different lines of code within one single program, were representing those different parts of the phenomenon. In this sense, students were having different lines within one single routine, to correspond to different parts of the phenomenon or their program that they talked about.

Second and related, typing long programs, takes time and in addition you cannot run them unless you have completed the program. The latter is a structural characteristic of programming in MW: a program in MW has to start with a “to” and ends with an “end” in order to run. Students usually started from the beginning and continued until they finished all the code for their program. A third possibility is that the beginning of a program in MW has to include several lines of code that are not important for what the program does, but are needed for the program to run. That code is similar to all the programs in MW: students need to identify the turtle that the code is for, setup its shape and size and if necessary create variables to be used in the program.

In this sense, writing new code is a rather technical activity, which has little to do with developing models, in addition to adding the "stress" of writing a program that is bugs-free. Probably because of this, students' conversations were limited during the time, and their focus was on the code it self.

Running and debugging new code

Create a button

As soon as a first draft program was ready to be tested, students switched windows (category "switch windows" on figure 3.9 represents such switch from the program window to the simulation window and vise versa) and created a button for their new program. Creating a button is basically a shortcut for running a program in MW, and it is done by assigning a button to a particular program. In this way, a simple mouse click initiates that program.

Creating buttons was probably a way of helping students to move quickly through this phase of their work, given the fact that during debugging, they usually ran their programs frequently. On the other hand, however, it is not clear what would have happened if it were harder to run their programs during debugging: would students still run them many times or would they take their time and read through their programs trying to find any bugs?

Creating buttons had another interesting role, this time methodological. In most cases, the act of creating a button in MW was an indication of a new code that students had just finished writing, which I used during analysis to identify what students were doing. In addition, students in all groups did not usually create a button prior to writing their program. This was possibly because students needed to type the program's name in the button, which they could not do successfully unless they had written that program.

Switch back and forth between the program and the simulation windows

During debugging, students went back and forth between the program and simulation window, ran their program, read any possible feedback provided by the software, identified bugs in their code and then fixed them. This pattern is shown in figure 3.9 below. (Note that the dots represented in figure 3.9 are not equivalent to the amount of time that a particular activity took place. During parts without any conversation, work is represented by a single dot.)

In general, throughout program strategies I, students were code-oriented. They were typing code, trying and debugging it as their goal was to create a program that would run successfully. Of course their ultimate goal was to create a simulation of a natural phenomenon, but a first important step was to create one that runs!

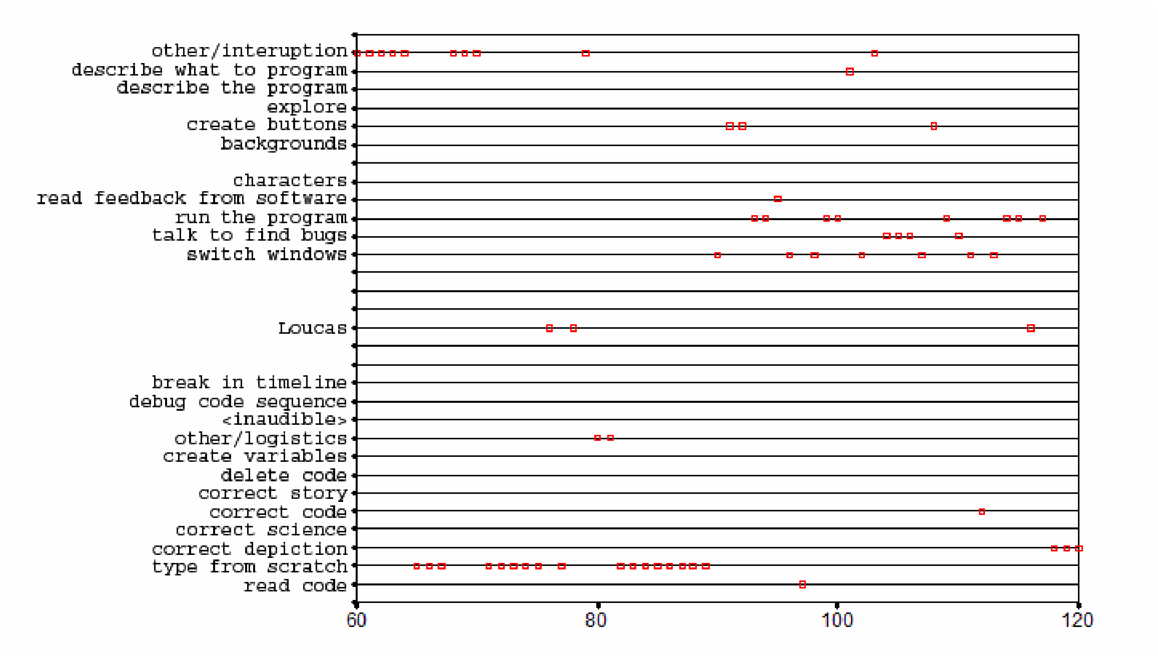


Figure 3.9. Program strategies I
(Source Aaron & Richard group, 13 November 2002)

Episode MW5, Conversation while programming, 13 November 2002, Richard & Aaron

This episode is an excerpt from Richard and Aaron's work. For the first time, in line 33 Aaron started writing their first line of their program. For most of the conversation that I present below, Aaron was the group's "typist".

33. **Richard:** to…
34. **Aaron:** enter, then talkto…
35. <silence while clicking/typing/looking on the screen> (Richard leaves the group)
36. **Loucas:** ok, I want to help you to….let me see. Oh!
37. **Aaron:** here's fd 1 wait 2 fd (Richard back in the group)
38. **Loucas:** so, make to jump moon and to jump earth.
39. **Aaron:** ok. (Richard is back at the group)
40. **Richard:** change to jump to moon, to jump moon.
41. **Aaron:** ok, this, yea, this was the moon, to jump, this is moon.
42. <silence while typing> (Richard leaves the group)
43. **Aaron:** … fd
44. <silence while clicking/typing/looking on the screen> (Richard back in the group)
45. **Aaron:** …bk 2 wait …bk 2 wait
46. **Richard:** isn't it 2.5
47. **Aaron:** yes, it's 2.5.
48. <silence while typing >
49. **Aaron:** .5 wait…. All right, control F, make a little button…jump moon
50. <silence while typing>
51. **Richard:** jump moon, jump moon!
52. **Aaron:** ok, hey!

53. **Richard:** I don't know how to move (reading from the feedback trying to run the program)
54. <silence while clicking/typing/looking on the screen> (Richard leaves the group) (Richard back in the group)
55. **Aaron:** hey, why is it so tiny?
56. **Richard:** what, what, control V (he presses control F)
57. **Aaron:** control V?
58. **Richard:** jump moon. Edit copy paste. No, no, they can't be spaces. They can't be spaces
59. **Aaron:** oh, ok, I get it.
60. **Richard:** you need to delete the button and you need to delete the thing <inaudible>
61. <silence while clicking/typing/looking on the screen> (Richard leaves the group) (Aaron clicks the mouse about 20 times probably trying to run the program which does not) (Richard back in the group)
62. **Richard:** Aaron, go back (presses control- F, Richard shows Aaron something on the screen possibly the space between jump_moon) backspace
63. <silence while clicking/typing/looking on the screen> Aaron presses Control F and tries the program again.
64. **Richard:** he is jumping!… we made him jump. Mr. Loucas Louca! (Richard leaves the group) we made him jump.
65. **Loucas:** ok, I am coming!
66. **Aaron:** he's jumping! Yea! Let's go!
67. **Richard:** go back and change it a bit. After fd 2…
68. **Aaron:** yea make it a bit more ….the first one is fd 2…2 wait 4. So we need to double all of that.
69. **Richard:** wait 3, wait 3.
70. **Aaron:** co, cause see, we double this so we double that…(Richard leaves the group) so now fd … no,
71. <silence while clicking/typing/looking on the screen>
72. **Aaron:** all right. You were right Richard, we should do it fd um wait 3 (Richard back in the group) and this is fd 3 (Richard leaves the group) wait 4, now
73. <silence while clicking/typing/looking on the screen>

In the above conversation, Aaron typed their first program for an astronaut to jump on the moon. During this time, from a total of 41 utterances, 10 of them indicate no conversation between Aaron and Richard (and they were usually longer than the other utterances). Richard left the group 6 times and returned shortly to see what Aaron had typed. Aaron made all the typing and one possible interpretation of Richard's action was to provide Aaron with "the space and quiet" to finish typing up their program.

Students' conversations were technical, mostly about the code that Aaron was writing, typos he made, and possible bugs. Aaron quickly typed their first program and tested it. Then he returned several times to the program window to identify the bugs that were preventing their program from running. Changes during that time were only meant to make the program run, without paying any attention to e.g., the structure of their program, on which they seemed to be focused on during planning.

Surprisingly, as soon as their program was successfully running, Richard asked Aaron to go back in their code. They then talked about making some changes, mostly in the amounts of forward and wait instructions. In line 68 and 70, it started sounding like Aaron was thinking about the program structure again, talking about keeping a particular pattern with their numbers in the two parts of their program. They had two parts in each jumping program: the first part was for the upwards motion and the second for the downward motion. They seemed to want the numbers in these parts of the programs to be consistent with each other. However, there was not much justification of why this should be (other than being consistent) and especially Richard did not pay much attention to the argument that Aaron started to make. That they had a program that was running successfully was enough to celebrate and call me to show me their simulation.

3.3.3. Program strategies II: Correcting depiction

As soon as the students had a program that ran smoothly, they started correcting how the simulation looked. This seemed not to be a difficult transition from program strategies I, and it usually occurred as soon as students had a program that ran successfully. A few minutes earlier students were focused on the code and how to write up a program that ran successfully. Now they switched their focus on how to change their program to improve the simulation. In this sense, students used the code as a tool for improving depiction in their simulation.

I would like to pause here and note that the ultimate goal in this research study was to investigate ways, like the ones I am reporting here, that students used CPEs in the context of writing programs that created simulations presenting natural phenomena. One of my expectations during this study was to see students using the code to construct programs that would include the mechanism of what is actually causing the phenomenon. Prior to this study, I was not sure about the different possible ways that students can or tend to use CPEs as modeling media. It is possible to suggest that there is more than one way that students can use the code: during program strategies I students were using the code to create a simulation that runs. In program strategies II students were using the code to change their simulation to depict more accurately how the phenomenon looked.

Within program strategies II there were two distinct major activities: correcting depiction of their simulation and doing some fine tuning in their programs, playing around with mostly numbers and to see possible results in the simulation. In both cases similar activity patterns were isolated, as shown in the figures 3.9 and 3.10, and the difference in the purpose of their work was identified from the transcript of their conversations.

Correcting Depiction

To correct the depiction of their simulation, students were making changes in their code to change what the simulation was showing. Their focus was on the simulation itself and their goal was to change the code in such a way that would create a better simulation. Most of this work was based on trial and error techniques, since the code of their programs sometimes did not reflect their program plans (see episode MW6 and figure 3.11). Students were making changes to their programs and testing them, trying to create the best possible simulation.

Fine tuning

In fine tuning, students had a program that was running ok with no major problems in depiction. They “played around” with the numbers in their program, trying to improve the simulation. For example, Aaron and Samir had two programs that simulated an astronaut jumping on the moon and a boy jumping on the Earth. Their first program resulted a simulation that was not showing jumping higher on the moon than on the Earth, as they wanted. When they fixed that (corrected the depiction), Richard changed a few numbers in their programs so that the differences could be seen more clearly in the simulations (fine tuning).

Conversations

Another characteristic of program strategies II is that students started becoming more vocal in their interactions and started having conversations about their programs. Now that the concern of getting their program to run was gone, students were starting to have conversations about their simulation. Their focus during their conversations as described in part in conversation type IV was on how their simulation looked and how to change it to result a more accurate or appropriate simulation: that is, a simulation that would look better. Because of that, students also started using references to experiences they had in their conversations, supported their ideas and suggested changes for the code of their program. The simulation triggered conversations that would include episodes of students looking around in their experiences for instances where their ideas were easily found and observed, in an effort to convince others about the changes they were suggesting. During this time however, the use of code was minimal, and used to indicate only how to make those suggested changes. Even though the changes that students were talking about had to be done in code, students would only talk about the simulation and what was the phenomenon like.

Still, during this part of their work, students did not talk about the mechanism that was causing the phenomenon, which could have supported their discussion about changing how the simulation looked. Students were talking about the simulation, and in particular how the simulation looked. This kind of conversation did not necessarily need any reference to the code, and because of that students did not use the code or use it to talk for the suggested changes.

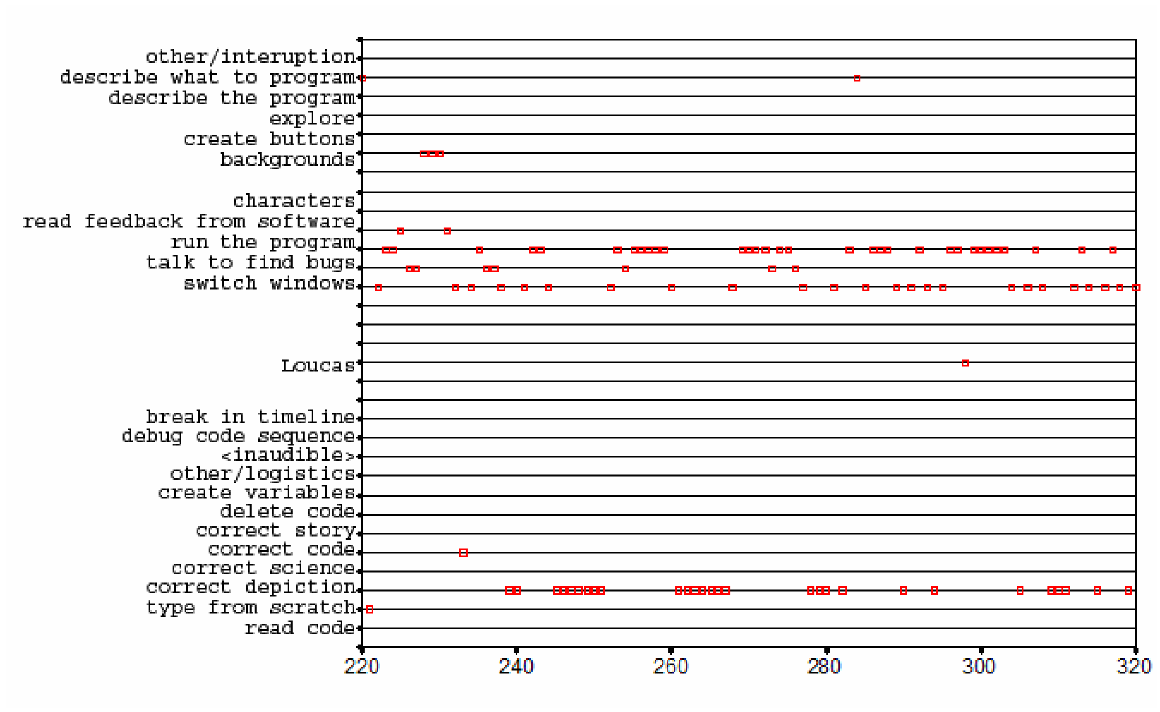


Figure 3.10. Program strategies II
(Source Joe & Samir, 13 November 2002)

Episode MW6, Correcting depiction, 13 November 2002, Joe & Samir

This episode starts after Joe and Samir had successfully debugged their first program. Now that they had a program that worked, they started making changes, mostly concerned with how the simulation looked.

213.**Samir:** that was way too many repeats, that was….
214.**Joe:** do it again, do it again!
215.<laughter>
216.**Samir**: we have way too many repeats.
217.<silence while clicking/typing/looking on the screen>
218.**Samir:** I know what we have to do, we have to do seth 90.
219.<silence while clicking/typing/looking on the screen> - Joe left the group
220.**Samir:** I got it!
221.**Richard:** hey, you got it,
222.**Samir:** well, it's <inaudible>
223.**Richard:** hey Joe look what Samir got.
224.**Samir:** heeeeeey!, I got it somewhat!
225.**Richard:** you got it down, look, Joe, look!
226.<silence while clicking/typing/looking on the screen>

In the above conversation, students were focused on changing their program in such a way that the simulation would look better. In the conversation excerpts above and below, students were concerned about how their simulation looked; they were making small changes mostly to numbers, (amounts of forward, waits, directions etc) and then they tried them out. They were not concerned about their program or their program structure.

Samir continued for another 10 minutes to work on these changes, while almost the rest of the class had a conversation about another's group program. Samir decided to stay and "fix" their program because the arrow was "not working too well" (line 228). Richard would occasionally visit Samir and they would briefly exchange a few ideas – but still Samir was the typist and he was not very vocal.

227.**Loucas:** Samir do you want to come here?

228.**Samir:** I am trying to make the arrow to move, it's not working too well.
229.<silence while clicking/typing/looking on the screen>
230.**Samir:** Oh, I missed!
231.**Richard:** put it like there with the arrow.
232.**Samir:** well, that wouldn't look well, I need to <inaudible>
233.<silence while clicking/typing/looking on the screen>
234.**Samir:** look, watch!
235.**Richard:** it's going up!
236.**Samir:** I guess it's going up like 50, see like <inaudible>
237.<silence while clicking/typing/looking on the screen>
238.**Richard:** make it go down then, go right, go left.
239.**Samir:** I should repeat a few more times.
240.<silence while clicking/typing/looking on the screen>

After they fixed their program, I had the opportunity to have a look at it. When I first approach the group, Joe and Samir showed me the simulation. It looked ok, mostly what they talked about before: the arrow traveling first upwards, then horizontal for a while and then downwards, stopping on the target. Their code, however, did not reflect any of their ideas. As figure 3.11b shows, they had two separate repeat lines, possibly representing the structure of two programs, one for the upward motion and another for the downward motion. In addition, even if they set the angle of the direction of the arrow to be 45 degrees to begin with, they used seth 90 instead, which sets the direction of the arrow to be horizontal, turning it for 0.5 degrees left for the half of the motion and right 0.5 degrees for the other half of the motion.

As important, their first program departed significantly from at least one of the ways they were thinking during planning, which is resented in figure 3.11a,c,e. (Joe and Samir talked about two possible plans, one being about a program that would have several smaller subroutines, and another in which the arrow would follow a half oval trajectory, as presented in FIGURES 11a,c,e). Their first program may have been resulted a similar simulation with what they were thinking during planning, but the code of their programs was significantly different. When later asked about this, Samir suggested that he was aware that their program was different from their plans and that reading the code (figure 3.11d) resulted something different than what their simulation was showing. However, he indicated that this was the only way that they could make that simulation look ok.

As the conversation continued, I asked Joe and Samir about the code of their program, because I was not sure why they had abandoned their initial ideas. It almost seemed as they had started thinking about this in a variety of useful ways, but they sacrificed everything for how their simulation looked.

279. **Loucas:** so Joe, come here. I don’t understand this.
280. **Samir:** we did, like we think it did, see when we had the same it just looked like it was doing this <gestures>. So I had to make this a bit higher for do like this <gestures>. I just didn’t seem to like this. So I changed it to <inaudible>. The angle, it’s going up and <inaudible> found it and changed it to <inaudible>.
281. **Loucas:** Joe, Joe I want you to explain because I can’t understand your program.
282. **Samir:** well, I wrote this.
283. **Loucas:** but, so, so tell me. So you have local angle, and then you make that 45 degrees because you want to start 45
284. **Samir:** u ha.
285. **Loucas:** and then what happens?
286. **Samir:** and then…
287. **Joe:** then…
288. **Samir:** then repeats, then repeats by 30, as angles going up by 0.5, 0.5, 0.5 …
289. **Loucas:** so, but you are not this angle.
290. **Samir:** and that would be <inaudible> then we can just <inaudible> that angle out.
291. <silence while clicking/typing/looking on the screen>
292. **Samir:** or we can just make this we can take that out and make it 45, 45.
293. **Loucas:** ok, I, so, I was thinking something else, that you were saying…
294. **Samir:** I know, I completely understand what I was doing, but that’s the only way that worked out. I tried to make it the way we had it, and it did like this. <gestures> the arrow went like this.

In line 294 Samir indicated something that made the situation clearer. Indeed they had some initial ideas about how to start programming, but after trying them, this was “the only way that worked out.” And he continued: “I tried to make it the way we had it and it did like this …” indicating that their ideas had outputs that were not those that they expected.

The focus of the conversation and their work had shifted from the program structure (conversation type I), to the details of the code they would use (conversation type II), to write a program without bugs (program strategies I), and finally on how the simulation looked (program strategies II). Even though that before students were seen to think and talk about the structure of their programs, referring to the different "parts" of the phenomenon they were modeling, they were now focused on simply getting a simulation that would just show what happens, without any references to their initial plans.

There is, however, a possible contradiction for the above argument: one can possibly argue that students' focus was depiction all along. When they were concerned about the structure of the program, they suggested that "… we can make a bunch of little programs to make it, 'cause that [one single program] wouldn't be as realistic […] because the arrow would go hum, hum, hum, hum <gestures indicating different directions of the arrow>." (lines 77-80 from the whole group conversation on 13 November 2002). At that point students were referring to how the arrow traveling in the air looks, and were using that information to support their idea of writing several small programs. In the last part of the conversation, students were again concerned about how the phenomenon looks, but now they were using that to write a program that would create a simulation that would look like that.

It is possible, however, that talking about their future program may have been more useful for making progress in developing a model of the phenomenon than their first attempts to write a program. When they talked about program ideas, their focus was on the program structure, thinking about how to divide their programs in ways that made sense to them. Thinking about the structure of their programs was helpful partly because it helped students to break the phenomenon into small discrete programmable pieces that have similar characteristics. When, on the other hand, students started writing the code, they used trial and error techniques to write a program that would simply result in a simulation depicting the phenomenon.

An additional possibility is that students started with particular ideas about what and how to program. While typing code, they were also trying their code, a very easy way to make sure that their program worked. It is possible that because they were trying their program, they started thinking about how it looked (and not what the code represented about the phenomenon); they wanted to get their program to create a simulation that would look a particular way, without worrying about their code itself. In fact, the last 10 minutes of the video of that day, while talking with me, seemed to have captured students' effort to fit their program into a theory they were creating on the spot. They were using (fancy) words that they possibly heard before (e.g., momentum, line 325), without being clear how these were related with the situation.

322. **Loucas:** is that what happens in reality, it changes from an oval to…
323. **Joe:** I think yea, I think because um, <inaudible> because you <inaudible> straight up, but if you put the bow like that it will go at an oval shape, but then near the end it will start to go down like a circle shape.
324. **Samir:** yea because it's, it is after it runs out of the …. It's not like the <inaudible>
325. **Joe:** it runs out of momentum.
326. **Samir:** yea, it <inaudible> it goes really fast but then it starts to lose the power of <inaudible>

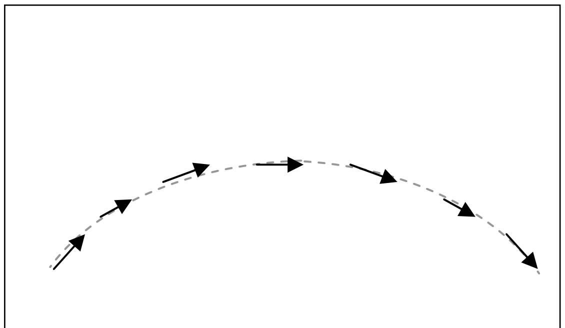
The difference in the fds of their two subroutines was due, as they indicate above, to the fact that the arrow at the second half of its trajectory runs out of "momentum", which maybe a possible explanation for the program, but it came up as an explanation only when they have created a program that looked ok (using trial and error) and had to justify it.

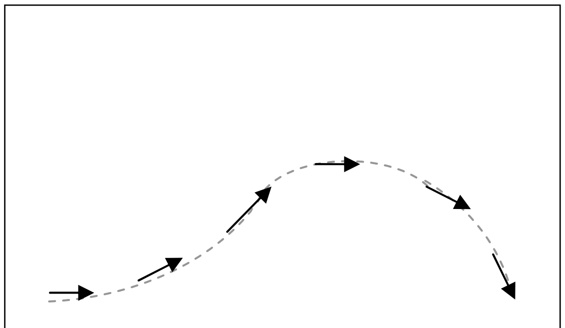
```
to shoot
talkto "a
local "angle
make "angle 45
repeat 90 [seth :angle fd 5 make "angle :angle +
1 wait 0.1]
end
```

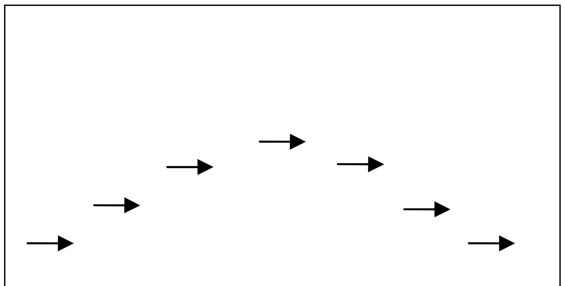
a. Joe & Samir’s (possible) proposed program

```
to shoot
talkto "a
seth 90
repeat 30 [fd 5 left 0.5 wait 0.1]
repeat 40 [fd 5 right 1 wait 0.1]
end
```

b. Joe & Samir’s first program


c. Their simulation plans


d. How their code reads


e. How their simulation would have looked on the screen

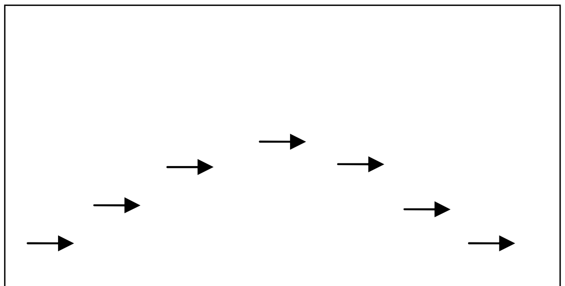

f. How their simulation looked on the screen

Figure 3.11: Comparison between Joe & Samir’s plans and first program[5]

[5] The simulation on the screen in MW is different from what actually the program reads because when turtles are replaced with characters such as an arrow, MW does not turn the picture of the characters when there is code saying e.g., right 90.

3.3.4. Program strategies III: Modifying code to change the science that represented

A third type of program strategies was characterized by students' efforts to make corrections to their code to represent correctly the phenomenon under study. This was different from prior activities that were focused on writing and debugging code to make their program to run. Rather, students' focus during program strategies III was on modifying the code to represent the phenomenon (in code) in addition to creating a simulation of the phenomenon.

Students' activities in program strategies III were different from program strategies II, where students were making corrections to the depiction of the simulation. In program strategies III, students' focus was on how the phenomenon was represented in code. For students to attend to the code and talk about how the code represented the phenomenon, they needed to read the code instead of simply running it, and in this way students started using the code to talk about the simulation. This is why this conversation was only observed in 2 groups out of the total 4 that were using the MW. Due to this, I only suggest that program strategies III shows a particular possible way of working with CPEs rather than a well documented emerging theme of the study.

I do not suggest that this type of programming is an emerging pattern because it was not observed in all groups. However, it indicates the potential use of the combination of code and simulation to trigger conversations about how natural phenomena are caused. That the students were indicating that the code was not ok, even though the simulation looked ok, is a possible indication of thinking about using the code to read how the phenomenon happens and possibly what might be causing it.

An important implication of this type of conversation is that this is probably a useful way of using CPEs for talking and developing models as representations of natural phenomena. In this way, students used the code of the simulation to talk about the natural phenomenon, and about the reasons why the code was not accurately representing the phenomenon. Students conversations during this type of work was about and with the code and in addition students were making references to experiences they had to support their ideas, comparing code with simulation results and real-life experiences they had about natural phenomena.

Episode MW7, Changing code to represent a new phenomenon, Joe &Nick, 4 December 2002

This episode describes the first 30 or so minutes of the meeting on the 4th of December 2002. The meeting started with Joe and Nick, since Samir was late. The conversation was about the program that Joe and Samir wrote during the previous sessions. Even though their program resulted in a simulation that looked ok, the code of their program was not in sync with what students seemed to have talked about during their planning session.

3. **Loucas:** here is what I see. Repeat 30 fd powers and then left 0.5. so… if it is left 0.5 then you start like that, and then you go like that, little bit, little bit like that. So you go left, left, left and then repeat 40 right. So then it's right, something like that.
4. **Joe:** yea, that's right, that's the, that's what it does.
5. **Loucas:** ok, so you wanted to do that, and you didn't what to do like a thing like that.
6. **Joe:** yea, we did want it to do that, but, um, like, actually I think what Samir um, like, control –F place, this what it does.
7. **Loucas:** I understand, I understand Joe that it looks ok, but I am asking…
8. **Joe:** no, wait. Let me to, um, here 10, place, …um see what it seems to make is the shape is kind of like that.
9. **Loucas:** ok. But if you look closely on your program, the program says that right? Because it's going left, left, left, and at some point it's going right, right, right. So if it's left it's going like that, and then … I understand that it looks like that, but the program actually does this, right?
10. **Nick:** but that looks ok.

11. **Loucas:** I understand that, but there are two different things: how it looks and what the program tells right?
12. **Nick:** but, well, what are, our purpose now is to show someone science, only, we are not showing then science by them going on this, to this thing (the code window) and…
13. **Loucas:** why not?
14. **Nick:** because when we're playing the games we can't go into that control-f!

My first attempt to have a conversation about the code had similar effects as before. When I draw their attention to the fact that their code said something different from what actually seemed to happen in the simulation, their response was "But that looks ok!" (line 10); their focus was on what the simulation looked like (line 4 & 6) and not on what one could read from their code (line 12). Students seemed to use MW as depiction medium, as a means that creates a simulation that shows how reality looked. During that part of their work, their focus was on what they saw on the simulation screen and whether that looked ok. Code in this case was only used to make the simulation look as good as possible.

Nick, however, indicated something interesting. Even though he seemed to be aware of the ability to read the code and figure out what the program did, instead of just running the simulation, he indicated that there was no point for that: "when we're playing the games we can't go into that control-f!" (line 14) meaning that the code was used to write a program that creates the simulation, but after that is done, the code is basically useless for the user.

To spark a conversation around the code, I suggested students to write a program that would represent throwing a rock, hoping that the differences in the code between the two programs (arrow vs. rock) would help students talk about what their code was representing. Instead of writing new code, students decided to copy the existing code and modify it to fit the new phenomenon. This sparked a new kind of conversation! Nick carefully read the program and indicated that "this program isn't what an arrow does!" (line 28).

28. **Nick:** no, actually this program isn't what an arrow does! But anyway. An arrow actually, wait, sorry, but…
29. **Loucas:** hold on. What's that?
30. **Joe:** that throwing a rock it would make a shape.
31. **Nick:** ok, that's what a rock does. What the program is doing that would what a rock does. This is what an arrow does. An arrow drops just like a gun bullet does! A gun, like when you shoot a gun, the bullet would drop.

The new discussion had two new characteristics. First, with Nick's contribution the focus on the discussion was on the code itself. Nick indicated that the first part of their program (that resulted in the upward motion of the arrow in the air) was not what it should have been. Things that are shot straight (like the arrow here – since the code indicate a horizontal direction at the beginning of the program – and a gun bullet) continue to move straight and "drop a little" (line 31). Second, Nick initiated a conversation in which he made references to experiences from other situations that were more clear: the case of a gun bullet was probably for Nick a clearer situation of what happen in cases that things are thrown straight in the air: they move horizontally and fall towards the ground.

32. **Loucas:** that's not what happens here though!
33. **Nick:** yea, but that's what a real arrow does.
34. **Joe:** no, no, not really, a real arrow usually goes like that or something.
35. **Nick:** no! I've seen one.
36. **Joe:** yea, actually…
37. **Nick:** look, instead of going, they go like this.
38. **Joe:** and the arrow is just goes like that?
39. **Nick:** yea, and when it drop a little bit and then it hit.
40. **Joe:** …but amazing.
41. **Nick:** its power

Now students started talking again about parts of their program, but the focus was on the behavior of objects in their simulation, as this was reflected in the code. They were talking about how the actual motion should look and what the arrow would do during its motion. I need, however, to note that "how the phenomenon looked" was still part of their conversations, even though, they were using it to support their ideas about how the code should have to be. The focus, though, was on the code.

An interesting point about this conversation is that 18 lines before line 28 (in line 10) Nick suggested that the simulation looked ok. Now, in lines 28, 31 and 45, he suggested that this was not what an arrow was/should be doing. Most probably, the change in his thinking occurred due to his reading of the code – the simulation itself was not helpful in really thinking what the arrow was doing.

45. **Nick:** it's not what an arrow does! Unless you're going like this <gestures>, but the <inaudible> is going like this.
46. **Loucas:** but Joe, but why (Nick) do you say that? Because Joe is saying this is what happens.
47. **Joe:** Well, I mean. Yea, we can try and do that. But...
48. **Nick:** yea, but what I think if you want to make it more realistic then you can drop. You should drop doing a little bit up.
49. **Joe:** well, we could do that, but …
50. **Nick:** cause, cause that would make it look more realistic but if you wanted to, if you wanted to look realistic going like that, then make it go like this and then it, and then it will go…

Nick and Joe spent several minutes in changing the copied program according to their new ideas. The focus of their work during that time was changing the code of their simulation so that it represented more accurately the new phenomenon. In this group and in others, too, this occurred after students had an initial program that they debugged, and had a conversation about what it represented. The interesting thing here is that Nick and Joe wanted to write a new program that would be similar to the one they had, but different in the science that it represented. Instead of going through the process of writing new code, debugging it, then possibly fixing the depiction represented by the program (that is what usually students did), they started with the program that it was running ok and showing ok, to change the science it represented.

Among the characteristics of having simply to modify new code is the speed of having a final product, that the focus is now on the science that the program represents and not on the code itself or on how the simulation looks and also the fact that students interact during their work. As indicated before (see section 3.1.4 & 3.2.2) during typing new code, interactions between students are very limited, with only one student typing and the other simply watching, leaving the group and coming back. While modifying their code, however, students' interactions were increased (in this case about 50 utterance of a total 2 minutes of work!)

3.4. Summary of MW findings

Contextual analysis and analysis of student conversation revealed several different types of conversation and activities for students working with MW. During planning of their work (conversation type I), students working with MW mostly talked about the structure of their program, breaking down the phenomenon they wanted to represent in small programmable pieces that share common characteristics in the way the phenomenon looked. During that time, students used the program language as their communication medium and they talked about how their simulation would look in order to support their program decisions. In conversations II, students talked about their program details and how to program specific ideas. During writing and debugging their programs, however, students working with MW focused on writing programs. Conversation type III, which occurred during writing and debugging, and included limited interactions between students, which were strictly technical, i.e., about the code and program primitives. In conversation type IV, students talked about how their simulation looked, making references to everyday experiences, while debating whether their simulation was “realistic.” Lastly, in conversation type V, students described what happened in the in their simulations, using code as a tool for creating their simulation.

Early in their work with MW, students dealt with the characters (objects) and the background(s) of their designs. In program strategies I, students were observed to write and debug new code, while having limited conversations and trying to get their program to run successfully without paying any attention to their program structure that they had talked during planning their work. Program strategies II, include student efforts to correct depiction of their simulation, fine tuning details of their simulations, and were starting to have more verbal interactions about their work. I classified program strategies III as efforts to modify code to change the science that representing, seeing the program as a representation of the phenomenon. In this sense, modifying code however, to match, i.e. a slightly different phenomenon or idea, seemed to be a more productive context for modeling in science. Students were reading their program in detail and identifying what each line of code represented, in order to make appropriate modifications. In this way, students did not have to deal with any technical issues to make their program run – but, they had to think of ways to represent the phenomenon in code.

4. STAGECAST CREATOR FINDINGS: CONTEXTUAL INQUIRY & ANALYSIS OF STUDENT CONVERSATIONS

Contextual analysis of students' work with SC revealed several common types of students' activities and conversations. Similar types have been identified in all three groups whose work was analyzed and unless otherwise noted, examples of types of students' activities and communications were found in all groups.

Below, in separate sections I present conversation types (combinations of conversation patterns that were common among groups that I analyzed) and activity types (combinations of activity patterns). Conversation types (CT) include CT I: Talking about the overall story line of their games, CT II: talking about the overall story in front of the computers, CT III: Translating the story into code/rules, and CT IV: Talking while programming. Program strategies (PS) (activity types) include students activities while dealing with characters and backgrounds of their designs, PS I: Making and changing programs and PS II: "Reading" and modifying rules.

4.1. Types of conversations

Findings from contextual inquiry and analysis of student conversation revealed several different types of conversation that students had in different parts of their work with SC. Below I present and discuss the different conversation types, providing examples of student episodes.

4.1.1. Conversation type I: Talking about the overall story line of their games

The first type of conversation that was common among the groups occurred in the context of talking in whole class, talking about future work. This was a common theme that I ran across every time I presented a new situation to students and asked for possible ways for programming. It is important to note that this type of conversation followed planning sessions that children had every time they were starting to work on a new design, and thus it is possible to suggest that their presentation also reflects their planning work.

When presenting their ideas for programming, students were talking about two things. Firstly, they usually talked about the setting of their game/simulation that included possible background, characters and possible details about the characters' behaviors. Secondly, students were talking about the overall story of what they were planning to represent with SC, describing in details "scenes" of their story which would happen one after another in their simulations. For instance, when we had a conversation about a ball falling off a cliff, students would describe what is going to happen to the ball at the beginning, how the ball was then going to have a faster speed, thirdly how that speed was going to increase once more and so on, almost describing the motion of the ball frame by frame. Unless I provided them access to a computer, students did not talk about ways to program such motion. In a different group, students were talking about creating a game (see episode SC1) in which balloons full of helium would fly with different speeds in the air and a boy would shoot them down with darts, gaining points for each balloon that he would shoot. It almost seemed that students were describing the overall story of what was going to happen in their designs: I decided to refer to this as the scenario of their designs.

Story details may not be useful for programming per se

Students were also very descriptive about the details of their scenario that might have been not useful for programming their ideas per se. Being descriptive about details of the overall story is more useful for making better and more “realistic” games/simulations, simply because these details are probably important characteristics of video games. These details included the kinds of the objects to use in their games, their colors, etc. However, in many cases, students were not talking about the natural phenomena in their designs, such as why and how different balloons (with different amount of helium) travel with different speeds or how the darts for shooting the balloons would travel after leaving the boy’s hand. A requirement for their work with SC in the second phase of the study was to include in their designs things that happen in real life and are related with science. Despite the fact that students were aware that for instance the above two physical phenomena were parts of their designs, they did not talk about details that were related with how that phenomena would happen and how they were going to program them in their designs.

Viewing their work as creating games

Students working with SC seemed to view their work not as programming *per se*, but rather as creating games. Programming in SC is the process of creating rules for assigning behaviors to the objects of the simulation. Students were not concerned about the development of particular rules, even though it was the only tool they could use to develop their simulation. They seemed to view their work with SC as a process of creating games, and this was reflected in their work during planning when they talked about their plans for programming.

Viewing their work with SC as the process of creating games, at least for the early parts of their work, students were dealing with the details of their games rather than thinking about the details of creating the program/rules that would create their game/simulation. This seemed to add the difficulty of having to translate all the details about the scenario of their game to programmable rules in SC (see discussion later about their work during programming).

There is, however, a potential advantage of such thinking. Thinking about a series of sequential events, which make a story, might be a productive way of analyzing natural phenomena, breaking them in pieces that are meaningful for understanding and possibly studying. This is in fact a known way of studying molecular phenomena in biology, such as a series of events caused by one protein, which leads to the production of another protein that causes an action (or reaction) leading to a change of a third protein and so on. Even though thinking for natural phenomena (in the context of representing them with SC) in such a way might not include any information about what is actually causing the phenomenon, this might be a way that could possibly lead to, for example, a conversation about combining scenes of the story in a rule that creates different scenes instead of simply presenting each one independently. That is, rather that having three rules representing three subsequent scenes of the falling ball, each one with a different speed, students could have a single rule that would cause the change of the speed during the motion of the ball. However, at this point of their work, students did not use any of their ideas about their scenario to talk about how to program them.

Episode SC1, Presenting ideas to whole group, Annie & Bryan, 18 November 2002

Episode SC1 took place during the second phase of the study. During the previous meeting, students spent their time brainstorming ideas about their final project. The only requirement was that their programs should include phenomena of science that happen in everyday life. Each group had some time to think and talk about their ideas and how to put them in a program. Students spent the meeting brainstorming ideas, looking at other students' programs related to science and mathematics (that were available on SC website), and talking about whether their ideas were appropriate and easy to program. During that meeting, students agreed that their programs would implement some kind of a game that would include ideas from science.

The transcript below starts after students had some time to talk in more details about their plans for their game. The excerpt is taken from Annie and Bryan's presentation.

51. **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…
52. **Bryan**: for example like the less point there, the bigger balloons are.
53. **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.
54. **Loucas**: what do you mean by wipe-outs?
55. **Annie**: like if…
56. **Jeremy**: deleted!
57. **Annie**: it's like a <inaudible>
58. **Bryan**: no, you're on a balloon and then, you are on a balloon and then you like flowing up with a balloon except like if you pop into a greater balloon it will pop and fall back again.
59. **Annie**: but if you, if you, if you like we're gonna have pots at the bottom, and you can <inaudible> and aim it somehow, anything should be below<inaudible> The gray balloon the all your points will just be deleted.

Thus far students talked about different details of the overall story of their game. They presented them mostly in a visual manner, describing the story of their game and giving details about different characters (balloons) in their game. They were very descriptive and as the conversation continued provided more and more details about aspects such as the colors of the balloons, the size of the balloons and the speed of the balloons, because those characteristics were important for their game: the “gold” balloon worth more points, ought to be faster and if possible smaller so that it was going to be difficult to shoot it down.

Being focused on the overall story of their game, students were not talking about the parts in their game that were related with science (which in the previous meeting they had discussed). Neither did they talk about how to implement all their ideas about their game into code, a conversation that required talking about rules and how to translate an idea (e.g., a faster balloon) into a rule in SC. As the conversation continued, I asked to talk about their idea of the helium being the cause of the balloon’s speed.

12. **Loucas**: ok. Ok, ok. Um, are you thinking of making, are you thinking of what we talked about like if a balloon has more helium it goes up faster, like this?
13. **Annie**: our balloons will go around the screen.
14. **Loucas**: but see, if a balloon, a balloon doesn’t, doesn’t go back and forth, it only go up, goes up. Right?
15. **Bryan**: yea, that’s what, yea, that’s what we were saying.
16. **Annie**: but it wouldn’t go in straight path exactly.
17. **Bryan**: see, it will be like in the air and it will go like <inaudible> and it couldn’t go up with the balloon, ‘cause then it could just like, let’s say there’s a gold balloon which worths more points and then if you’re going up with it will be easy to shoot at that, if you only stay at one position and then it, you can’t <inaudible>
18. **Loucas**: where are you going to put the person that shoots balloon? Where he or she’s going to be?
19. **Bryan**: <inaudible> she is going to be in the sky, and there are gonna be this giant ball of balloon or something.
20. **Jeremy**: oh my God!
21. **Bryan**: well I said there might be!
22. **Seth**: he should be on the side.
23. **Loucas**: so Bryan let’s, let’s focus on, on one balloon, let’s say one balloon, let’s take the red one, ok? What’s, how do you program that balloon? So you have a balloon. What do you want to do with that balloon?
24. **Bryan**: well, it says, if you go to the right one of those moving <inaudible> things, it says at the bottom, it says that slow, fast, like it says like slow, kind of slow, fast, very fast. We could like program the balloons and see like for make a gold balloon if we can put like first very fast, but not, I mean fast but not really fast.

25. **Loucas**: right, I know what you mean, you want to make each balloon to go fast, or faster…
26. **Bryan**: or slower
27. **Loucas**: …depending on something, right? But that you're referring to you can't change for each balloon. It's like changing it for the whole game. So you can't make it faster or slow it with that. You want to make a balloon faster, there are other ways to make that ok? Remember like you made the ball coming down going faster? That way! Ok? So, we're, we're gonna work on that. Good.

At the end of the conversation, Bryan started talking about a possible way of programming the balloons' speed, even though he did not seem to be really interested about how to program their ideas. In addition, it was not clear whether Bryan was certain about how to program balloons with different speeds, and his idea in line 24 possibly indicates that Bryan and Annie did not talk about that before. Later in their work, Bryan would bring up this idea again, and Annie would disagree, because Bryan in line 24 was referring to the speed of the whole simulation and not to the speed of particular objects. It is possible that Bryan in line 24 was just thinking of that idea, simply in response to my question, and thus suggests that students did not think about those kinds of details (programming details) during planning. They only talked about story details.

4.1.2. Conversation type II: talking about the overall story in front of the computers

Conversation type II was a conversation among students and myself, and happened early during their work with computers. In conversation type II students were sitting in front of a computer and in some cases, including the case in episode SC2 below, students had started working with their designs. In all cases the conversation was prompted by asking students what they were doing or what they were planning to do.

Shift in context of work did not seem to cause a shift in early conversations

Despite the change in the context of their work (in conversation type I students were not sitting in front of a computer) students did not seem to have a much different conversation from before. Students talked about the simulation that they had already developed and about the pieces of their story that were left to be incorporated in the simulation. One could expect that, like students working with MW, the context of actually programming with CPEs, can shift students' focus to become more technical: students had now the new task of putting rule or code together to implement their plans. However, at this point of their work, students were still focused on their overall story that they wanted to represent or parts of it that they had already represented and wanted to modify or improve.

Talking mostly about their game scenario

During this type of conversation, students started to use SC's programming language. As figure 4.1 shows, however, talking with the PL was not as often as talking about the story to be programmed. It is possible to suggest, as illustrated and in episode SC2, that students were still in the "mode" of designing and talking about the scenario of their games and in some cases overwhelmed by the details they wanted to include. Programming language started to be used as a way of being more precise while talking about their game scenario, possibly using it as the tool to create their game/simulation and not as a tool to represent mechanisms that cause physical phenomena.

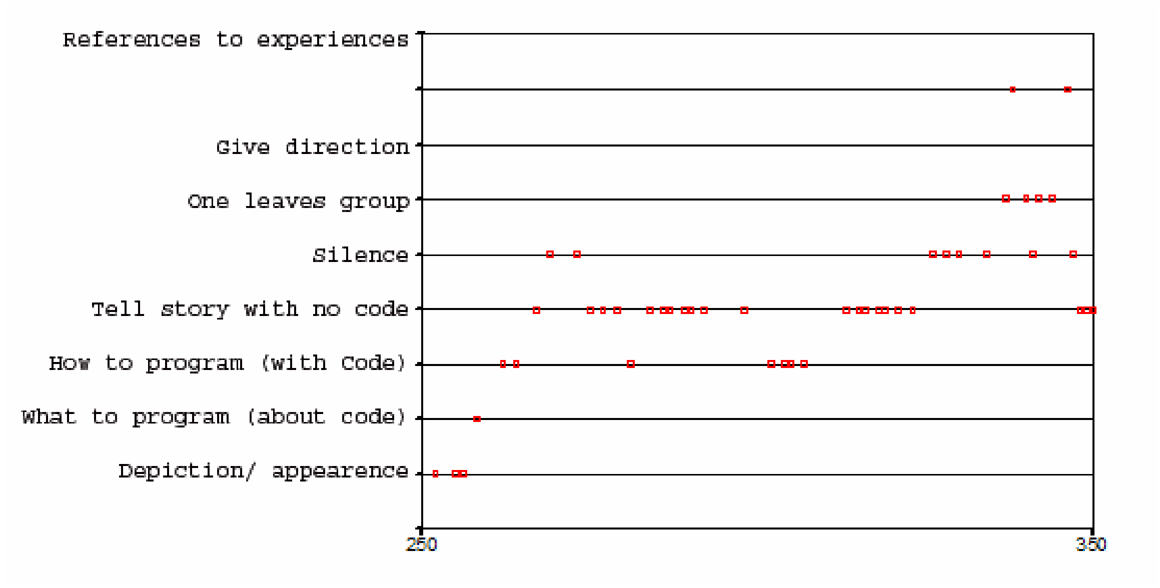


Figure 4.1. Conversation type II

Episode SC2, Talking about their simulation and their story, Annie & Bryan, 18 November 2002

This episode is an excerpt from the conversation from Annie and Bryan's group. They had just spent some time developing a few rules for balloons to fly with different speeds on the screen, and they wanted me to have a look at it. Their rules were simply descriptive of the behaviors of the balloons without any explanation included about how different speeds were caused. Their focus, however, was on the simulation and how that looked and when I jointed their group, they first ran their simulation for me.

245.**Annie**: so, let's, let's get Mr. Loucas to look at this.
246.**Bryan**: ok. Mr. Loucas can you look at this?
247.**Annie**: ok plus, press play. The gold is fastest, then blue, then red.
248.**Loucas**: ok.
249.**Annie**: but, I'm gonna have more of these colors, but for now…
250.**Loucas**: so, ok, yea. So stop it. Let's, let's so, what, what do you need to do next?
251.**Bryan**: we just, we just.
252.**Annie**: it goes up and stops, it goes up and stops, it goes up and stops.
253.**Bryan**: we should, we should.
254.**Loucas**: ok, here it is. Here there is something running, moving at the same time. So now it should look…
255.**Bryan**: oh, that's so cool!
256.**Annie**: yea, it's perfect.
257.**Loucas**: ok, we're getting better.

258. **Annie**: but wait, if we add more balloons with the same color, will the same rule apply to them?
259. **Loucas**: yep!
260. **Bryan**: yea! Cool!
261. **Loucas**: so, another question. What still needs to be changed?
262. **Bryan**: first we need to make the character, like a kid.

During the above 18 utterances, I asked them twice what needs to be changed in their design, hoping to get into some conversation about things that were not depicting physical phenomena (balloons traveling up, reaching the top of the computer screen, disappear and re-appear at the bottom of the screen) or about ways to improve the rules they had, in order to include the amount of helium causing the different speeds of the different balloons, as they suggested earlier.

In both cases, their responses were directly related with the scenario of their game and how their simulation looked. Annie in line 252 identified a depicting detail that changing would result the simulation running more smoothly, focusing on details that had to do with how their game looked. Bryan in line 262 talked about adding another character for collecting the balloons that would fall, after they were shot down, focusing on keep working to add the other parts of their game scenario.

Despite their focus on depiction, and using the simulation to create a game, Annie and Bryan did not seem to be bothered by the motion of the balloons in their simulation which disappeared from the top and reappeared from the bottom. From a science point of view this was a (depicting) “malfunction”; balloons should either disappear from the screen (i.e., keep flying up) or stay on the top of the screen if there was a roof. From a game point of view, this was not a problem; not everything in games has to have a tangible scientific mechanism that can explain them, as long as they fit the game scenario and have nice graphics.

As the conversation continued, I brought this to their attention, hoping to be a starting point for thinking about physical phenomena and representing things that occur in everyday life.

263. **Loucas**: here is what I see. I see balloons going up, and then coming from the bottom?
264. **Annie**: oh we need to stop, stop that, we need to stop, delete themselves at the top.
265. **Loucas**: ok
266. **Bryan**: yea, that's, um …
267. **Annie**: but then how do we on going if the levels are going on?
268. **Loucas**: I know what we can do, I'm going back to the screen. We can, …
269. <silence while clicking/typing/looking on the screen>
270. **Loucas:** Oh, I think it's on "stages"
271. **Annie**: no.
272. **Loucas**: yea, yea, yea. See? If I turn this off,…
273. <silence while clicking/typing/looking on the screen>
274. **Loucas:** <inaudible>
275. **Bryan**: yea, the person can't move any more, like so, the person will always stays in one place. It can't like go up and it can't…
276. **Loucas**: right, right, and he can… so you can make the person like being on a mountain, being like here, so he's gonna shoot the balloons going up, he can't shoot them up there.
277. **Annie**: what would happen if they're all filling and then like he's …..

Instead of helping them to start thinking about what was causing things or at least whether things that they had in their simulation happened in real life, talking about making a change with balloons, got Annie and Bryan worried about their game: how this change would affect their game story? Annie for instance indicated that she was worried about what would happen if the balloons started piling up at the top of the screen (line 277).

278. **Loucas**: ooooooh, ok. So what, so what we can do is we can put either doors up there like, magic doors that take the balloons somewhere else, or…
279. **Bryan**: that one would be, that wouldn't be um science.
280. **Loucas**: ok
281. **Bryan**: we can make like the balloons pop, or be, or be deleted. Because…
282. **Annie**: or…
283. **Loucas**: because they're off the sky.
284. **Bryan**: yea, they are out, well,

285. **Annie**: or we can
286. **Bryan**: it can still, it can still be, like in the ozone layer but still pop because they pressure.
287. **Annie**: I have an idea. After they get to the top, like we can have a bird or a character, regularly swimming around and as we have the farmer collecting the eggs, like the bird can go past the balloons and then would…
288. **Loucas**: oh, there would be a bird flying, and the bird pops them?
289. **Annie**: yea, and it regular flies across the top
290. **Bryan**: yea like it has <inaudible>

[…]

298. **Annie**: you have to make, it would make sense for it to go regularly around and pop balloons, and then there wouldn't be at the top any more.

An acceptable solution was to add a character of a bird that would fly back and forth on the top of the screen and pop the balloons that were "hanging in" there. Bryan suggested to have the balloons pop when they reach the top of the screen, and supported his idea by providing a possible explanation that the balloons would pop because they would reach the ozone layer in the sky and the air pressure would pop them (line 286). Annie did not seem to take Bryan's idea into consideration and she suggested adding the bird character.

Interestingly Bryan's suggestion included an idea from science (air pressure pops the balloons), even though his idea was far from being accurate. However, it seemed that the "idea from science" simply fit their game scenario and provided an easy way out of a situation that students did not like in their game. In this sense, students continued to be focused on their game scenario, making sure that each detail was figured out.

I wanted to try once more to get them to talk about causality instead of simply describing what happens in their game. It seemed that students were not talking about "science" because at that point of their work it did not seem to be meaningful: why talk about what was causing the balloons to fly in different speeds, when it is easy to assign them with a simple rule that does just that without any need for explanation? However, students could possibly talk about mechanism by talking about the story of the balloon which loses helium because the bird made a hole in it. Losing helium could have been part of their scenario, but at the same time it was a mechanism that could account for the change in the balloon's behavior: at some point it would cause the balloon to start falling towards the ground

308. **Loucas**: Bryan, what if we do now, what if you, instead of having different rules of this kind [causing the balloons to move in different speed based on their color], what if you say, you know, the gold balloon has 50 helium. And as of a more of the helium it has, the faster it goes.
309. **Bryan**: right.
310. **Loucas**: right? So that when, when the bird pops it, will start, will not like pop, it will just get a hole loosing helium,
311. **Annie**: oh yea.
312. **Loucas**: so be coming right down, so the other person will have a second chance to shoot it, that balloon. What do you think about that?
313. **Bryan**: and then the, um the like points, it cuts in half…
314. **Annie**: oh, oh!
315. **Bryan**: because like, it's like helium inside it coming down.
316. **Annie**: but we could just have like a whole rule so that it could say anyone put down, but what it would happen when it will make it to bottom, like a girl would run across.
317. **Loucas**: they could stay on the bottom.
318. **Bryan**: they'd just be on the bottom.
319. **Annie**: but they will pile up!

Once again, Annie and Bryan were focused on the overall story and whether the new addition to their game would fit with the rest of their game scenario. Annie suggested that the balloons would pile up at the bottom, unless a character ran across the bottom of the screen and picked them up. Bryan, on the other hand, was thinking about how balloons with holes losing helium would affect their scoring system for their game.

Students were very precise about details of their story, such as what would happen to the balloons going up and then coming down, how to avoid them piling up on the top or on the bottom of the screen. Despite that, students did not refer to any details of how they were going to create rules that would result in such a story that they were talking about. Their focus was to have a complete story, with all the characters and details in place.

Because of that, as I discuss later in students' activity patterns, when they started programming their game, *students working with SC were focused on the whole*, that is on the "system" that they were designing and not on individual objects and their characteristics and behaviors, as well as on *what was happening* instead of how that was happening. Focused on the whole, Annie and Bryan wanted a program that would result in a game with different colored balloons moving in the air with different speeds. The mechanism of the balloons' motion, and in more general the mechanism that actually causes the different speeds in each balloon was not their concern. In addition, they were interested to show what was actually happening in their design and not necessarily to reflect how their rules created particular behaviors (blue balloon moved slow) and system characteristics (e.g., a combination of three balloons moving with different speeds created a particular pattern on the screen) . If depiction criteria were met, there was no further need for any other kind of rule or any conversation about existing rules.

4.1.3. Conversation type III: Translating the story into code/rules

In this third type of conversation, students' focus shifted to talking about the ideas to be programmed using the program language. Their conversation became more detailed: students started talking about how to implement their plans using SC's programming capabilities.

These conversations usually occurred well after students started working on a new design. This might have been a possible shift in the context in which the conversation occurred. The most important difference in the context in which conversation type III took place is that students had already developed a number of rules for a number of characters (objects) in their simulations. Now that students had spent some time programming, when talking about future work they seemed to talk about what rules to make and what behavior each rule to cause.

Focused on programming

During conversation type III students were more focused on programming. As figure 4.2 shows, their conversations were mostly coded as exchanges of talking with the program language and talking about their scenario, in an effort to translate their ideas into rules/code to add in the simulation. In this sense, students were talking about one idea, and then talk about how to write a rule for representing that idea in the simulation. This was an important characteristic of their conversations and it was different from talking about how the phenomenon looks. Their focus before was on the scenario that they were about to use in their programs. Here, in this third type of conversation, most of their discussion about how to use SC's program capabilities, probably suggesting students' efforts to identify the most appropriate way of representing their characters' behaviors. The scenario of their game was still their concern, but their conversations seemed to become more technical and focused was now on what kinds of rules to create.

Using the program language

An important characteristic of conversation type III is that students used the program language. Students seemed to use the program language for two purposes: (1) to tell their story, possibly utilizing the SC's program language as a communication medium to share and communicate their ideas, and (2) to be precise. As a communication medium, program language can be more precise because it is used to develop instructions for the objects to follow in the program. In this sense, a conversation about a disagreement about the motion of a dart (see episode SC3 for details) may better be supported by the use of the program language, which can provide students with the clear ways of talking about their ideas. In the case of a dart thrown horizontally, different speeds of dropping while the dart moves, are important issues related to the mechanism that causes the dropping to happen in the first place. The use of program language may help students to have a productive conversation about the actual mechanism that causes the phenomenon.

Making references to experiences

A second characteristic of this type of conversation is that students started making references to experiences (this was not observed in the previous types of conversation), referring mostly to known everyday experiences in order to convince each other about how to program specific ideas in their games. Interestingly this mostly happened in the context of debating about particular details of the rules (such as how many squares should the dart fall towards the ground, or how many squares should the speed of a falling ball change etc).

Now that students started debating about the details of their rules, they had to support their ideas. To do so, they turned to experiences from their everyday lives, to support their ideas with related phenomena that they observed daily. This might have been the beginning of a conversation about the mechanism that caused the phenomenon under study. Differences in the behavior of objects, discussed in the context of talking about particular programming ideas, may or could have led to a conversation about what actually is causing the phenomenon.

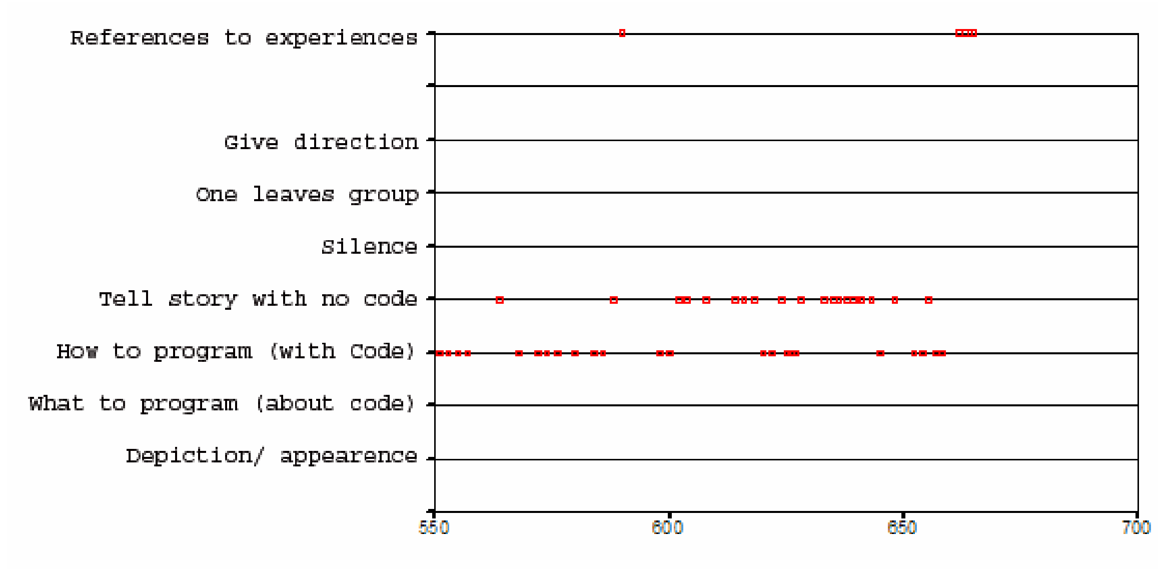


Figure 4.2. Conversation type II

Episode SC3, Debating program ideas, Annie & Bryan, 25 November 2002

This conversation took place during the last 20 minutes of our meeting on the 25th of November. The excerpt below is taken from a conversation that I had with Bryan and Annie about the motion of the darts that a character in their game would throw in order to pop balloons and gain points. The conversation focused on programming the motion of the darts, a topic that students did not talk about before.

184.**Bryan:** ok, the character has like darts, like some darts in her pocket and the balloon start rises and then she got to aim at a balloon and then throw. But she can't … if you , if you expect the dart to hit the balloon right when it's aligned with you, that won't happen because you obviously miss because the balloon will still going up.
185.**Loucas:** ok.
186.**Bryan:** can I see the balloon for a sec? Let's say, here is the dart like this, when it comes up, something like this, and you aim right when in ….
187.**Annie:** but then again, if you're going that slow, it has a chance of getting it, if going like this, then it (the dart) starts it has a chance of making it. You shouldn't…well, the dart, the dart is really, really tricky.
188.**Loucas:** no, so let's make the dart. So the dart, if you throw out, how would it go? Will it keep going on a straight line?
189.**Bryan:** it will drop faster like after 7 squares.
190.**Loucas:** why do you think that?
191.**Bryan:** because if a dart it won't stand, it won't stand like at first it will go, yea, it first will go (showing with his hands straight) and <inaudible>

The first to talk about the motion of the darts was Bryan, who in a “story-telling mode” talked about how darts were part of their game scenario. After adding a character for a dart in SC, Bryan started talking about its motion. He suggested that after seven squares of traveling horizontally, the dart would start falling (line 189) in addition to keep moving horizontally. As I had seen before, Bryan was still talking about their story scenario, but this time, he was talking about the story of the dart, starting to talk in detail about its behavior.

Two different things in the above discussion are important to highlight. First, Bryan started talking about individual objects. He was still talking about the story of those objects but he was using the program language to refer to the details of the objects’ behavior: the dart would “drop faster like after seven squares” referring now to the motion of the dart. This was probably a useful thing to be happening. Even though it is possible to suggest that Bryan was still just telling a story, rather than focusing on the overall story of the game, he was now telling the story of a particular object. He was also referring to details of the objects’ behaviors (changes in the motion of the dart) for which it might have been at least useful to use the program language for communication.

Second, as Bryan suggested (line 186), the dart’s motion could be getting in the way of their game story, making it more complicated because, and as Annie indicated (line 187), the user should shoot the dart(s) well before the balloon was aligned with her in order to get it. From the perspective of science, students might have started attending to some physical mechanism that would explain the motion. The actual motion of the dart, however, seemed to interfere with the scenario of their game and at least in part, it might had been counter-productive for their progress with this program.

206. **Bryan:** it [the dart] will go like this. <showing on the screen that the dart moves on a straight line for a few grid squares and then start falling down>
207. <silence while clicking/typing/looking on the screen>
208. **Annie:** no, but that’s unusual.
209. **Bryan:** It’s not like going straight out.
210. **Loucas:** what do you mean Annie?
211. **Annie:** it actually, it wouldn’t; it just look realistic if it went straight and then move like that.
212. **Loucas:** so, what she, what should it look like.
213. **Annie:** I think that the idea that going diagonally, I think that it would be exactly in straight line <gestures indicating with her hands that the line would look downwards but should be straight line> it just, it had to show these squares <inaudible>

As the conversation continued, Annie disagreed with what Bryan was suggesting, specifically about the motion of the dart independent of their story, indicating that Bryan was showing a rather unusual motion for the dart (line 208). When I asked Annie what she meant by unusual, she indicated that she was concerned about the arrow going first straight and then start dropping. She suggested that the dart should move diagonally from the beginning. Figure 4.3 shows their different ideas.

For the next 50 utterances, Annie and Bryan continued to debate between the two ideas, trying to state clearly their ideas and talk about which of the two ways was making sense to represent the motion of the dart. Most of their effort was to show clearly to each other the differences between their ideas. In line 274, Bryan started arguing against Annie’s idea, indicating that if they used Annie’s ideas, the balloons that would be on the other side of the screen would be difficult to be shot down, due to the distance from the shooter. Annie was quick to indicate that that concern should not affect how they would program the dart, even though Bryan seemed not to be satisfied with that.

Even though it was not clear whether both students were thinking in terms of the particular motion of the arrow, without being concerned about their overall story of their game, they seemed to have started attending to a discussion about the mechanism that was causing the motion. They started referring to everyday experience to support their arguments about the characteristics of the motion, they were using the program language to be precise about their ideas, and debated about an idea being “unusual” for representing a motion. However, it is important to indicate that neither the conversation continued to become a conversation about what actually was causing the motion, nor both ideas were scientifically accurate. They just seemed to be changing focus, from the overall story to the story of individual objects.

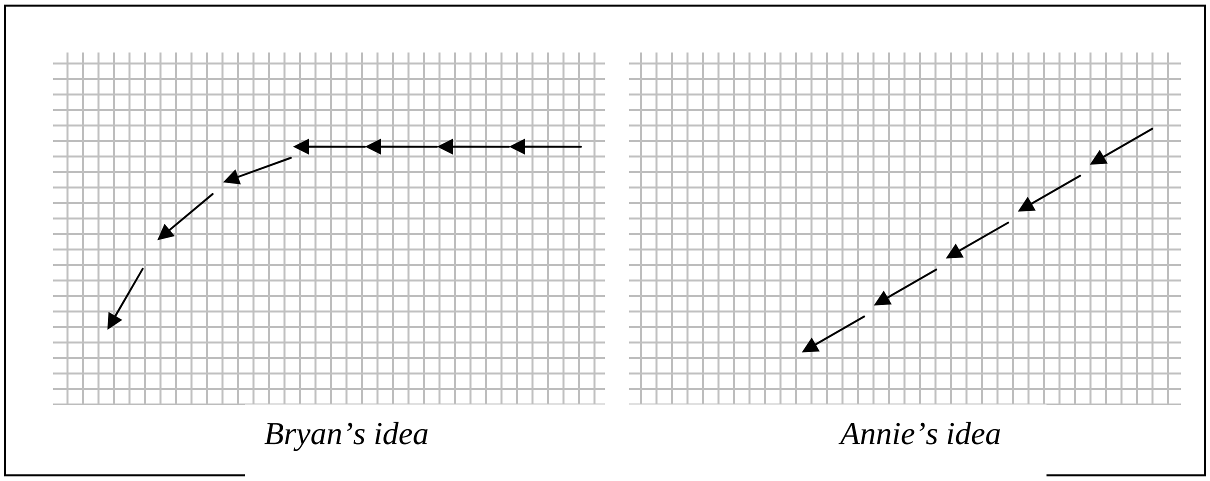


Figure 4.3. Bryan’s and Annie’s idea about the motion of the dart

Episode SC4, Two different conversation modes, Zen & Seth, 18 November 2002

This episode presents a conversation I had with Zen and Seth on the 18th of November about their program. Zen and Seth were making a game in which two people would compete in a race. Before I jointed their group, Zen and Seth were dealing with characters and backgrounds for their game and they had also created one rule for one of their characters to run.

This episode illustrates two different conversational types (II and III) which occurred within a few minutes one from each other. It shows the shift from one type of conversation to the other, as well as a shift in the focus of the conversation. When the conversation began, students started talking about their game, referring to details of their story, without being specific about the program details. After a short prompt, students started talking about developing rules for their characters to move and lose energy because of moving, providing details about how their rules would look and what to include.

155. **Zen**: um our bottle 1 is gonna be full of Gatorade. And that's gonna depend, depending on how much Gatorade you have in your runner is how fast they go. BUT, for every like certain amount of vertical they'll lose.
156. **Seth**: horizontal!
157. **Zen**:…horizontals they lose Gatorade, power. And so…
158. Loucas: ok. So, if a person stops, would he, would he or she be, um, getting energy up?
159. **Seth**: yea, yea, if you stop
160. **Zen**, no, no, no, but if they, if you couldn't, I don't.
161. **Seth**: we can't just control them.
162. **Zen**: I want to make it so that you can control them, like, because I want to be like , there is a long, long race and when it get close to like the end of the stage, it'll move, automatically move the whole stage over, you can do that I know! But, so like…
163. **Loucas**: yea, I know how you can do something different, but um along with those line…
164. **Zen**: yea, pretty much close, and so, you walk, you walk across and you see what's in front of you, after you are near like the end of the stage. And then so can we make it so that there is like, um,
165. **Seth**: make a <inaudible>
166. **Zen**: there is Gatorade bottles that are just kind aligned around
167. **Seth**: no, no, <inaudible>
168. **Zen**: you have to drive it or you fall down a hole or something. You make him drive across obstacles and if you hit an obstacle you like lose Gatorade power.
169. **Seth**: what I was thinking you've got Gatorade, is just like, it makes you, it makes you r power go up and it'll go up faster.

Zen and Seth were doing a nice job describing what they wanted to program. In those descriptions, they were using the program language, a possible indication that, in addition to thinking about *what to program*, they were thinking about *how to program* it. For instance in line 155 and 157 Zen indicated that for a certain amount of horizontal distance their runners move, they would lose power (which at the time was a variable that they were thinking of creating). However, in their descriptions they were also providing many details about different characters and parts of their story, possibly overwhelming themselves with the details of their story. In a way, they were just telling me a story, with almost too many things happening at the same time. In line 157 Zen talked about characters in the story to lose power when moving, in line 162 he talked about being able to control the characters (make them move using the keyboard, maybe), in line 164 he talked about having multiple stages that the race is going to take place, etc.

170. **Loucas**: so, let's, why don't you start with a person. Let's program a person to walk.
171. **Seth**: we already have one.
172. **Zen**: you HAD a person to walk.
173. **Seth**: get them back in, no, no, no, they're not in here.
174. **Zen**: yea, I just want to look, I just want to see if there is any people….
175. **Seth**: <inaudible>
176. **Loucas**: no, I put the person, the people are all in the other….
177. **Zen**: Yea, I don't want to use those people cause it doesn't look right with stage. Like… I just wanna, I mean, I guess we can do this one, but it just looks kind of…
178. **Loucas**: ok Zen don't worry about that. We can change the looks later. Let's focus on the person, the people now. So…
179. **Zen**: yea, let's change it back to the one we had. Ok.
180. **Seth**: we already have one
181. **Zen** oh, yea. What did I have, cats?
182. **Seth**: no, you had Milo, which is already there. We already had it in there, we're just putting it over and over.
    […]
188. **Zen**: I don't want those people.
189. **Loucas**: you don't want them? Why not?
190. **Zen**: just cause they're, I mean even though they have those like different positions so it looks like they're running, I just don't really want them, you know? Like kind of funnier people to look interesting….

Without any success, in the last 20 utterance I tried twice to make them start thinking about how to program their runners! Students were still focused on depicting details of their story. Zen did not like any of the characters available in SC to use for their race – and also they were unsuccessful to find new ones (lines 177 & 190). Even though in line 190 Zen did recognize that the characters that I prepared for them had several positions (that is different pictures that can be used to animate the characters using 2-3 simple rules in SC), he indicated that he wanted "funnier people" that look interesting. I was going to try once more, specifically to ask them to start with programming.

191. **Loucas**: sure. So let's program though these people and then we can change their shape and we can, we can import whatever picture you want.

192. **Zen**: yea!
193. **Loucas**: but I want you to do some programming.
194. **Zen**: so this guy knows how to walk right. He walks right, he walks 2 spaces…
195. **Seth**: he goes, we have to make his energy speed at 30 or something.
196. **Zen**: no their energy speed has to be like 5. No, no, no their energy speed should be like 10
197. **Seth**: 10
198. **Zen**: 10, yea, 10 and then you lose energy speed,
199. **Seth**: no we should make…wait, go down…
200. **Zen**: you lose 2 energy speed every time…
201. **Seth**: no he loses 3, the other guy loses maybe….

Now their conversation shifted to describing more specific, programmable details about their runners. They agreed that their starting power/energy was going to be 10 and they started to debate about the amount of energy that the faster runner was going to lose every time he walked 2 spaces. In this conversation, they were using "energy/speed" to refer to a variable that was basically an indication of the runner's energy. Gatorade, that came up in the conversation earlier, seemed to be connected with the runners' energy, because they seemed to think of Gatorade as the drink which would restore original levels of energy for their runners. As I found out later, they decided to name that variable energy/speed because the speed of their characters was based on that variable.

202. **Loucas**: so where, where every, your rule here, click on the rule, that is saying that it loses energy?
203. **Seth**: no, no we haven't said that yet.
204. **Loucas**: ok, so you need to do that.
205. **Seth**: click it twice, click it twice.
206. **Zen**: no, I mean is it gonna be another rule that says for every [vertical motion he loses energy], or that has to be the same rule [with the motion]?
207. **Seth**: no, no, no, that's yea….
208. **Loucas**: well if you're walking and you're loosing energy because you're walking, what do you think?
209. **Zen**: no I think it's not, he's loosing, loosing energy, he slows down and then he has to go again. But then, um I think that, I think it should be like you can <inaudible> walk but then the more like, the more Gatorade bottles you got the faster you can go. Or….

As the conversation continued, Zen asked me a question about a rule. Not only did they talk about how to program different characters' behaviors, but Zen asked whether a rule could cause two related behaviors: one being moving and the other being losing energy because of moving. This was a possible indication that Zen was thinking about the program language itself and not simply about how to implement his ideas with SC. In addition, the question was touching on one of the basic characteristics of SC. Rules in SC cannot have relationships between them (that is you cannot "call" a subroutine (rule) from within another rule, a technique frequently used in formal textual program languages). A way out of this was that you can create separate rules for the different behaviors and place them under a "do-all-and-continue" subroutine. In this case SC runs them both before moving to other rules. Another way (which was the way that Zen and Seth decided to follow) is to have both actions on the same rule: moving and losing energy.

Even though the conversation started to become technical, students were basically telling their story using the programming language. Like Annie and Bryan, the conversation continued with being about translating a story to rules so that it could be programmed in SC. In many cases that resulted in students talking about one idea (talk about their story) and then debating different ways of programming that idea (talk about what code to use).

4.1.4 Conversation type IV: Talking while programming

Students working with SC continued to talk and exchange ideas during programming too. However, other than talking during the process of creating rules, there was not a specific "type of conversation" in their communication during programming. During programming, however, students usually stayed in their groups and the process of creating rules did not seem to hamper their interaction. Unlike students working with

MW, students working with SC continued to talk during programming, in part debating about details of the rules they were making.

There were, of course, small pieces of conversations for different purposes such as short conversations similar to type II conversation, where students were trying to translate their story into program in SC. Also, there small pieces of conversations where students were giving directions to each other about particular ways of programming.

Students first talked about the part of the story that they would program and then suggested possible rules to assign to the characters that would result in that part of the story, like Seth and Zen in the following excerpt:

210. **Seth**: remember when you stop and the energy going up again? Remember? Remember?
211. **Zen**: yea.
212. **Seth**: So we have to do that.
213. <silence while clicking/typing/looking on the screen>
214. **Seth**: now we have to make it [the rule] when it's, when it [the runner's energy] gets to 0 it goes up. <inaudible>

Seth reminded Zen about making their runner to re-gain energy after he stopped and when Zen agreed, Seth indicated that they had to make a rule in which when the energy level is 0 (and the runner would stop running), the energy level would start going up.

During programming students also had conversations in which they gave direction to each other, about how to create rules for particular behaviors. This conversation was more technical than the previous conversation of translating a story to programmable rules, because students talking about particular actions (click here, drag this in this square etc) for developing rules, giving each other directions as where to click and what to type.

The following is an excerpt from a conversation between Seth and Zen on the 25th of November 2002, and happened during programming.

96. **Seth**: now we have to make that his energy goes down. No, make a rule, make a rule.
97. <silence while clicking/typing/looking on the screen>
98. **Seth** oh, yea, make a rule. <inaudible> move it down.
99. **Zen**: this gonna go…
100. **Seth**: wrong one, wrong one.
101. <silence while clicking/typing/looking on the screen>
102. **Seth** put 10. 'Cause remember?
103. **Zen**: 10.
104. **Seth**: remember he has more energy.
105. **Zen**: So? He doesn't stop as much.
106. **Seth**: oh, because he <inaudible>

[…]

112. **Zen**: now what?
113. **Seth**: oh, we did "and if.." wrong <inaudible> hit the other one.
114. <silence while clicking/typing/looking on the screen>
115. **Zen**: we need another variable.

[…]

152. **Seth**: no, make a new rule.
153. **Loucas**: why make a new rule?
154. **Zen**: because he has to <inaudible>
155. **Seth**: click that, double click that.
156. <silence while clicking/typing/looking on the screen>
157. **Seth** click that, click that.
158. **Zen**: it has to be, ok. This…
159. **Seth**: click that. Energy. Subtract, this <inaudible> what energy is.
160. **Zen**: from….
161. **Seth**: energy.

In the excerpt above, Zen was holding the mouse and Seth was providing Zen with directions of what and how to program based on the plans that they had previously discussed. In the conversation, Seth was providing step-by-step instructions of how to create rules for particular behaviors, and he was also showing to Zen parts of rules that they created which were "wrong" (line 113).

Generally speaking, in their conversations during programming, students were neither talking about different ideas to program nor debating about how a particular behavior should be programmed (how much did the speed of the ball change during motion, for instance). Rather their conversations were about translating their story into code and giving directions to each other for creating rules.

4.2. Program strategies

Contextual inquiry also focused on students' activity patterns during their work with SC. There were several combinations of activity patterns that were consistent among all three groups that were analyzed. These are presented and discussed below. They include students' activities while dealing with characters and backgrounds in their designs, creating and changing rules, and students' activities while "reading" and modifying their programs.

4.2.1 Dealing with characters and backgrounds of their designs.

When starting a new project with SC, students spent the first 20-30 minutes dealing with their designs' characters and backgrounds. During that time, their activities were coded mostly as finding and placing characters and backgrounds. Students were trying to identify those characters that would be more appropriate for their games. Figure 4.4 is an example from Annie and Bryan's group.

Figure 4.4 shows students actions during the first 20 minutes of their work with SC. As soon as students started working with SC, they spent the first 50 utterances browsing through characters. They briefly talked about their program plans, and about the characters that would have been appropriate for their design. For that purpose they "explored" different possibilities available through SC, looking specifically for characters and backgrounds for their program.

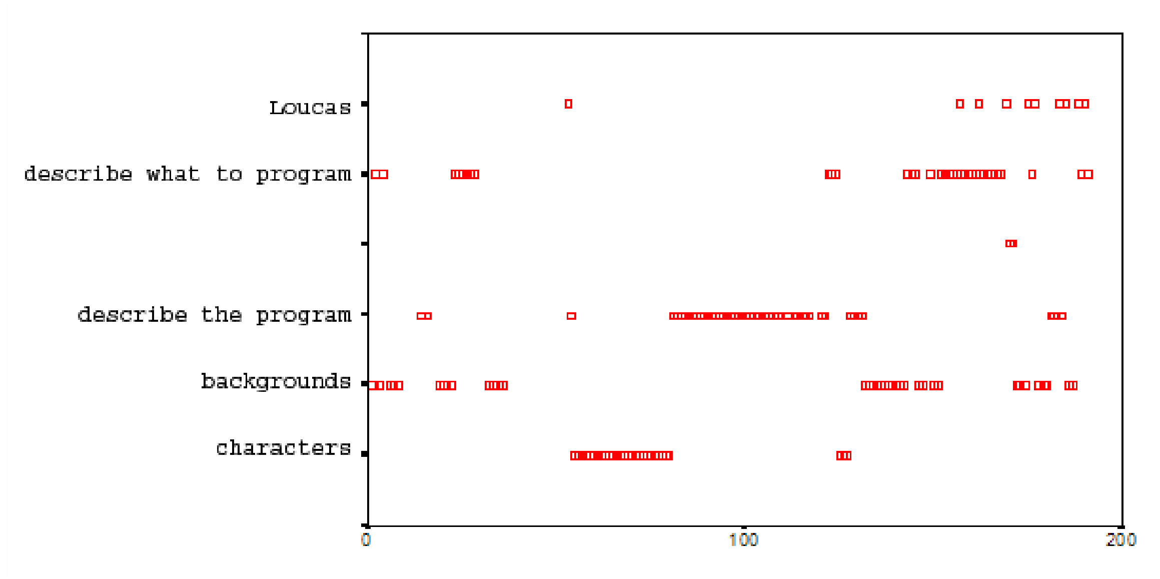


Figure 4.4. Students' activities in the beginning of their work

During the last 50 utterances presented in figure 4.4, I came to their group and asked them what they were doing. The conversation mostly revolved around their characters and backgrounds that they would like to use or started to use for their design. Students were talking about what characters they found and how they have modified their scenario based on the available characters in SC. They also talked about the part that each character would "play" in their game scenario, in addition to being detailed about how their characters and backgrounds looked.

Students' early work with SC had two important characteristics. First, students were only concerned about how their backgrounds and characters looked, dealing with depicting details of their characters and backgrounds. For instance, Zen and Seth were also looking around for characters for their race game, and were talking in detail about how their characters looked and whether they were appropriate for their game, as shown in the small excerpt below.

16. **Zen**: please, please, please, I'll, I'll, I'll just gonna, <inaudible> stages. Ok. There aren't any stages. Now, are there any specials?
17. (few seconds looking on computer without talking)
18. **Seth**: well, we don't need jar.
19. **Zen**: that's what I was thinking. Ok, now, characters,…
20. **Seth**: do they have different appearances?
21. **Zen**: I exactly….
22. **Seth**: he's way too big.
23. **Zen**: it's ok, one is bigger because he runs faster but has to slow down a lot.
24. **Seth**: which one?
25. **Zen**: ou! That's interesting. Ok. Smaller one goes slower, but he doesn't slow down as lot.
26. **Zen**: no, it has to be like a brown <inaudible> here. He's running.
27. **Seth**: we just say go and he trippers, and he slows.
28. **Zen:** no, no
29. Leaving from Stagecast, going to Internet Explorer, Google, to find characters for their design. They come back after about 10 minutes.
30. **Zen:** Ok, they're both stupids.
31. **Seth**: you once click on one guy.
32. **Zen**: ok. <laughter> ……
33. (few seconds looking on computer without talking)
34. **Zen:** oh, big boy can go…, whow! It's huge!

(excerpt taken from Zen & Seth group, 18 November 2002)

In the above excerpt, Zen and Seth were trying to find characters for their game, in which two boys would run in a race. In the first few lines (16-18) students were "exploring" SC's windows, in a search for stages and in line 20 they entered the characters' window. During the next few lines their conversation was about how the available characters looked (one was "way too big" – line 22 – and the other was smaller, but they were both "stupid" – line 30). They talked about details of the characters, that were not necessarily important about programming in SC, but they were rather important for their game and how it looked.

A difference with students working with MW is that students working with SC did not try to create a character from scratch, as some of the groups in MW did. Rather, students working with SC were willing to change their overall story in order to use the available characters to fit their story. It is possible that in creating games, visualization is a very important factor, because it is used to "tell" and "show" the overall story of the game. Despite the importance of the overall story to students working with SC (supported by their planning sessions and a big part of their work, as presented later in this chapter), and despite the rather user-friendly interface of SC (including tools for drawing characters) students were willing to change their story to fit the available characters. In one view, students were telling a story with their simulation, and depiction was an important part of that simulation.

A second important characteristics of students early work with SC, is that students wanted to make sure that they had all of the characters for their story in place (or at least most of them) before starting programming. After having them in place, students started creating rules for their behaviors. Unlike students working with MW, most of the students using SC did not return to deal with characters later in their work.

Characters were so important in their work, that during this exploration phase of their work, students would easily change their plans, due to what was readily available in SC. In fact many of their ideas in their stories derived from the software availability. Zen and Seth for instance talked about using a character of a bottle as Gatorade for their runners, to help them restore their lost energy.

4.2.2. Program strategies I: Creating and changing programs

Program strategies I, is a combination of activity patterns while students were creating rules for their games/designs. In the first type of program strategies, students were making rules, running their programs and then making changes in their original programs as a result of what they saw on the screen.

Creating rules

As with students working with MW, students working with SC were making rules and then they were trying each rule. The difference, however, with program strategies in SC is that students were running their program (rules) very frequently, at least more frequently than students working with MW. This might have been caused by the fact that rules in SC are independent programming units that SC can run individually. Every time students developed a new rule, they usually ran it to check it out. Figure 4.5 shows how this rule creation and testing looked.

That rules, however, are small independent units of programming in SC had a clear disadvantage. In all three groups that were analyzed for this study, students at some point were confused with the large numbers of rules that their programs included. Annie and Bryan for example could not understand why the fishes in their program could not eat other fishes when facing to the left, whereas they could eat other fishes when facing the right. At that point, their program included eight rules for the motion of the fishes (including four rules for diagonal movement) and another rule for eating other fish when facing to the left. Even though they tried going over their rules, they did not seem to be able to identify the missing rules. Similarly, in more that one occasion, Seth and Zen decided to simply delete all of their rules and reconstruct them from the beginning, instead of trying to figure out what was wrong or missing with their program (this happened while creating a program that had a ball falling, discussed in 6.2.2., and once while working on their race game)

Changes in the programs

Making changes in the programs was in most cases the process of deleting a rule and creating a new one. This was also an important characteristic of the students' work at that time, because they did not seem to try to identify what was wrong with their rules and to make changes in a rule. Instead, they preferred deleting a rule and re-writing it. Figure 4.5 shows the combination of activity patterns for program strategies I.

It is important to note that rules in SC cannot have syntactical errors, which can cause a bug and prevent the rule from running successfully. Due to the scaffolding that SC provides for programming, the only thing that can go wrong is to create a rule that causes a behavior that is different from the one intended.

Thus, an important issue is what a new rule looked like as opposed to the old rule that students deleted. As I discuss in episode SC5, in most of the cases students did not try to see what was wrong with the old rule (later in the chapter I provide some evidence from a single group in which students were reading and modifying their rules). However, early in their work, when their programs consisted of only a few rules that were relatively simple (did not include a lot of behaviors or any variables) students seemed to know what was wrong with particular rules but preferred to simply recreate them rather than edit them and make corrections.

This is not to suggest that students were deleting their rules without being sure what was wrong with them, because there is some evidence (that I provide below) of the contrary. At least in some cases students were talking about what was wrong with their programs (in some cases they specifically talked about particular rules) and simply instead of editing and making changes in their rules they deleted their rules and created new ones. Being accustomed to deleting rules instead of making changes in their rules, can become problematic for modeling. Instead of trying to *read* their programs, students may simply try to re-create rules without always being clear about what they were making differently.

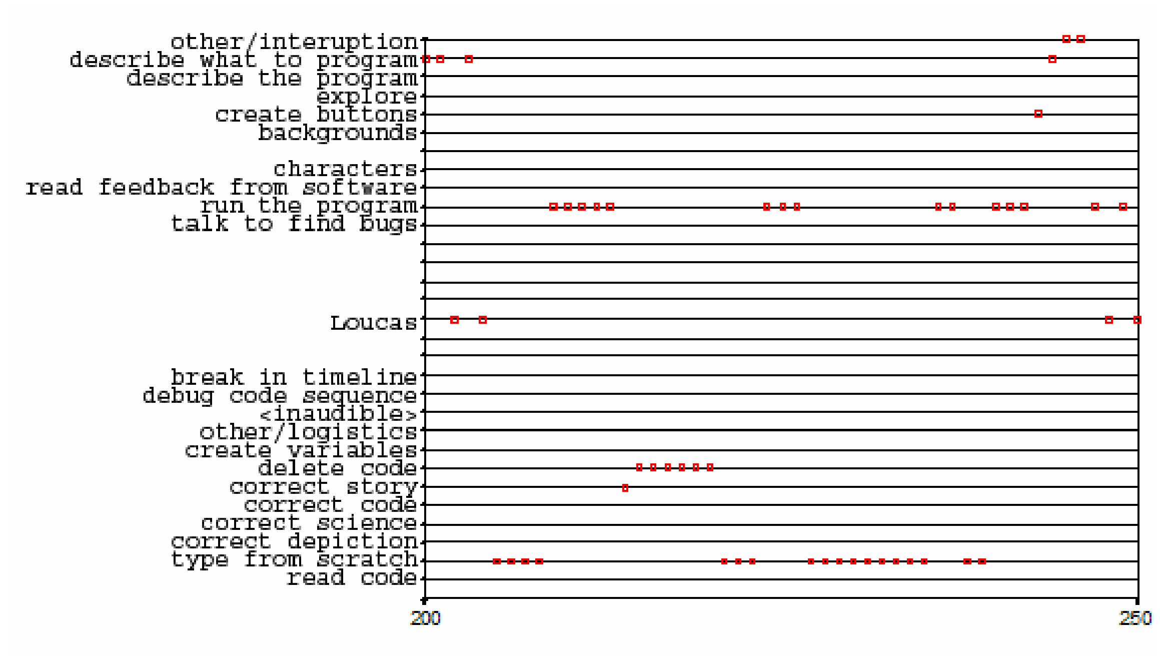


Figure 4.5. Program strategies I

In addition to writing rules and running their programs, students also had brief conversations about what was “wrong” with the rule they just created and what their actual intentions to program were, in an effort to identify what was wrong with their programs and fix them. This is represented in figure 4.6

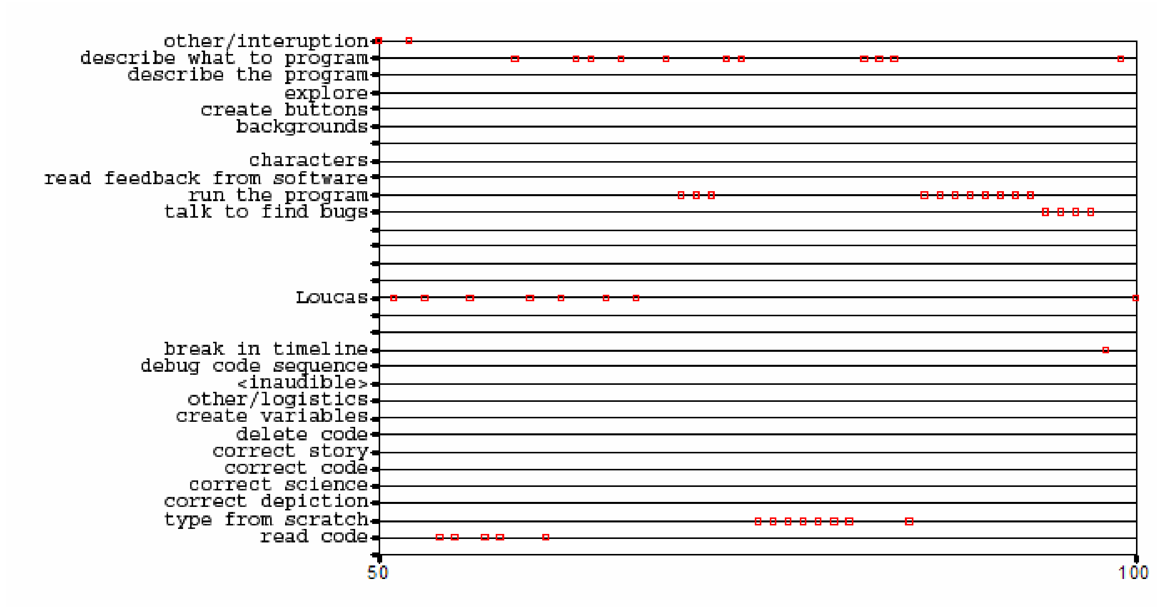


Figure 4.6. Program strategies I

Episode SC5, Program strategies I, Annie & Bryan, 18 November 2002

This episode took place at the beginning of Annie and Bryan's work with their new game. So far they were dealing with characters and backgrounds in their designs and in line 205 they started making rules.

205.**Annie**: all right. Let's make it move, um 2 boxes, at a time.
206.**Bryan**: like, why not 1?
207.**Annie**: cause that would be so easy to shoot.
208.<silence while clicking/typing/looking on the screen>
209.**Annie:** done. Now let's play. Right.
210.<silence while clicking/typing/looking on the screen>
211.**Annie:** stop! Ok, give me 3.
212.**Bryan**: No, that should be the gold.
213.**Annie**: yea. That's too fast.
214.**Bryan**: never mind, change the rule and make it one square, instead.
215.<silence while deleting the rule and making a new one>

Students began working with creating rules for the balloons. They first worked with the slow balloon, creating a rule to move the balloon two grid squares every machine cycle, because it would have been "too easy to shoot" (Annie, line 207). However, after trying their first rule, Annie and Bryan thought that it was too fast.

Despite Bryan's suggestion to change their rule, Annie deleted the rule and created a new one that had the balloon move slowly.

After that, they continued working with the blue balloon, making a rule for moving two squares at a time, and then continued with making a rule for the gold one. In every case they made a rule, and then tested it right away. While testing it, they talked about whether it looked ok (that is, was it too fast, or too slow?) and whether it would be ok for their game. For instance, in line 243, Annie indicated that it was just too easy to shoot their gold balloon, because it was moving slow. Bryan disagreed with her (line 244), indicating that how easy to shoot a balloon was not only based on the balloon's speed, but also on the trajectory of the dart that would shoot the balloon.

227. **Annie**: perfect. Now blue is next. It should go 2 squares. And then….
228. **Bryan**: 2 squares, but still slow.
229. **Annie**: yea, ok.
230. <silence while clicking/typing/looking on the screen>
231. **Annie:** I don't know this kind of square <singing a song>… and move 2
232. <silence while clicking/typing/looking on the screen>
233. **Bryan:** and then gold should just be one but make it fast.
234. **Annie**: and then how we would do that?
235. **Bryan**: oh, (showing the frame speed on the program)
236. **Annie**: that looks wrong. That cleared; it's pretty good.
237. **Bryan**: yea, I, I think so.
238. **Annie**: ok. Now let's do gold. Let's make gold 3. Just to see what it looks like and then if we don't like it we can delete the rule.
239. <silence while clicking/typing/looking on the screen>
240. **Annie**: ready?
241. **Bryan**: yea.
242. <silence while clicking/typing/looking on the screen>
243. **Annie**: why don't; we make it go a little bit faster? It's just less easy to shoot. Ok?
244. **Bryan**: um, ok. See like, it won't go like, it won't, like a dart will go like this. It will go like do, do, do, do so it may not be like that.

During program strategies I, students started to develop rules for their designs, focused on the overall story of their games. They were trying to create a simulation that was aligned with the story underlying their game, and in some cases they modified that story to fit the programming capabilities of SC. Despite the fact that programming in SC requires from the user to think about individual objects and their behaviors, students seemed to keep thinking in terms of the overall story and how rules were creating parts of their story.

In addition, during their early work with new designs (as the episode here shows) students did not have any conversation about what might have been causing the different phenomena in their game (e.g., what was causing the different speed for their balloons). Students were creating simple rules that created a simulation of how the phenomenon looked, and did not include in their rules anything about what the mechanism that was causing different objects' behaviors. In this sense, their models were simply descriptive of the phenomena they were representing.

4.2.3. Program strategies II: "Reading" and modifying rules

A second type of program strategies occurred later in only two of the groups (Zen & Seth and Sean & Tyra, could not be confirmed by findings in the third group). In those instances, students were not running their programs in order to identify flaws; rather, they started "reading" their rules and were having conversations about what their rules were doing and how should they change them. I do not use program strategies II as an emerging type of activities but rather as a possible way in which students (as partly seen in the study and) can use SC's tools for developing representations of natural phenomena. Figure 4.7 shows typing, deleting, reading and changing rules.

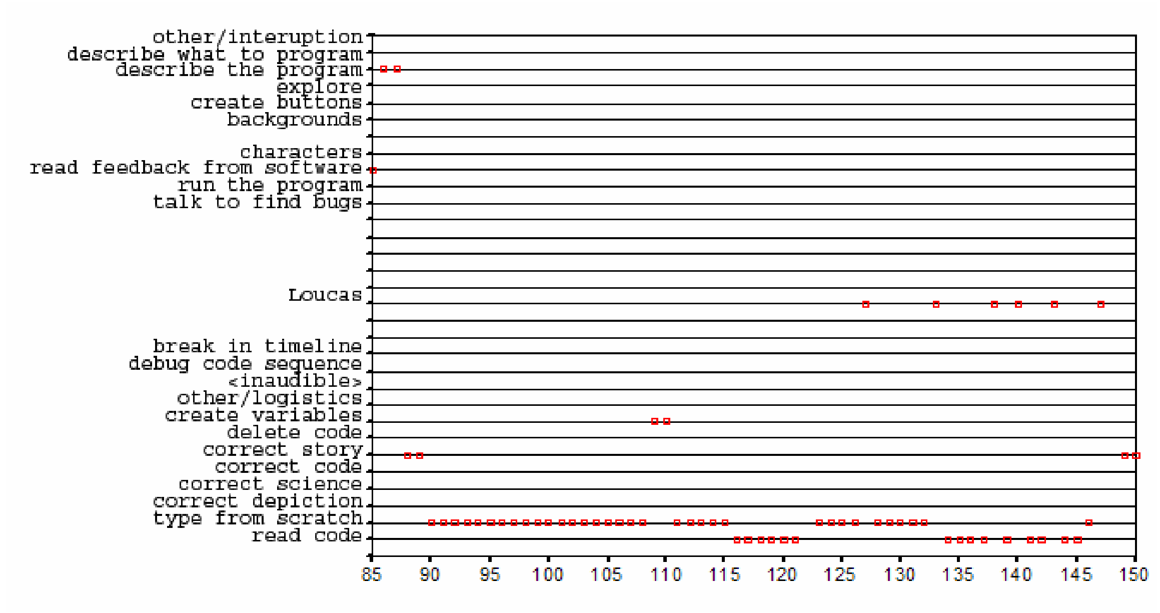


Figure 4.7. Program strategies III

That reading code did not occur in frequently, could suggest two possible things. First, as additional findings suggest (see discussion in the chapter 5), it might be difficult to read actual code in the form of rules in SC. Programming in SC is object-oriented. Code is written for particular objects and it is physically linked with the objects ("hidden" in windows that can only be seen if you double-click on characters). In addition, the activity of programming is a dynamic activity. Basically, the user needs to click on the program button and on the character that she is about to program, which launches a sub-application within SC. This takes the system into programming mode, which basically is a recording mode of changes. The programming mode is terminated when the user clicks on a "check" button. However, when the process of programming is done, programs are presented graphically to the user by beginning and ending states (Smith, Cypher, & Tesler, 2000). Thus, it seems that there is a gap between how programming is done and how it is represented – Smith (1994) refers to this as the programming-by-demonstration problem – whereas in traditional programming media, that is not the case: programming is done by typing, and reading the program, or just reading the already typed code. Graphical rewrite rules in SC serve as graphical reminders of the programs and they are not full representations. Rather they are hints of the rules (Myers, 1986). However, for debugging, the user needs to go to the appropriate character's window to locate the existing code. This is different from MW, where all the code for every object is located in the same window, where new code can be written or existing code can be changed.

Second, it is also possible that extensive experience with SC might help students overcome this problem of reading code in the forms of rules, as it possibly happened in this case, and as such students can debug their programs without necessarily running them. I highlight this finding because it is rather important for using programming in science. One of the advantages of programming is that one can read students' code and see how they represented the phenomenon. More importantly, students can read their own programs, which can possibly help them focus on the structure and mechanism that is causing the phenomenon, rather than the function and depiction of the simulation. In this case, by focusing on the code (or rules in this case) it might be much easier to focus on representing the causal mechanism of the phenomenon in the programs and not just writing rules that simply show what is happening.

4.3. Summary of SC findings

Contextual analysis and analysis of student conversation revealed several different types of conversation and activities for students working with SC. In planning their work, students working with SC talked in detail about the overall story of their games (conversation type I), describing a succession of events that would happen in sequence, one after another. That is the way they though of using SC, to represent a series of events, even in the cases of developing games. Their programs usually consisted of a large number of rules (as opposed to the single routine that early programs in MW consisted of) that were meant to be run by SC in sequence. In conversations II, students continued to talk about the story of their game, even thought they were sitting in front of their computers, ready to start developing their games. In conversation type III students were trying to translate their story into rules, were focused on programming and used the program language in their interactions. They were also making some references to experience to support their proposed ideas for their programs. Conversations type IV was coded as the conversations that students had while programming.

Early in their work with SC, students dealt with the characters (objects) and the background(s) of their designs, spending much more time with how their designs looked that the time that students working with MW did. In program strategies I, students were observed to create rules for their programs, with changes being made in the form of deleting rules and making new ones. During program strategies II students were reading and modifying rules of their programs. During creating and debugging their rules, students working with SC were focused in translating their story or game into rules that could be used to program a simulation. Their focus was on creating a simulation that would depict reality as well as possible, in addition to meet several criteria of good computer games (i.e. visualization, story plot, scoring system, levels of difficulty).

5. COMPARISON AND DISCUSSION OF FINDINGS FROM MW AND SC

The purpose of this chapter is to summarize and compare findings between students' work and their conversations with MW and SC, mostly from the second phase of this study. The purpose of this study was to investigate in detail the potential use of CPEs by young learners for modeling in science. The discussion below is based on the approach of using CPEs as modeling media in science. However, and as important, during the study, students worked within a framework of using CPEs to *represent* physical phenomena. In this sense, students were not using CPEs exclusively as modeling media. Students working with MW, for instance, tended to see their work as the process of writing sequential programs or representing phenomena with simulations. On the other hand students working with SC tended to see their work as developing games, and, at least in some cases, developing games was in conflict with development of models in science.

I also isolate and discuss the characteristics of CPEs that were supportive (or not) for collaborative modeling practices in science. The summary and comparison includes a discussion of several emerging themes about students' work with CPEs in science. I start with a comparison of students' activities and conversation with respect to their approaches to planning, writing and debugging code and using code as a representation of the phenomenon. In this discussion I summarize findings from the two previous chapters and highlight differences in students' activities and conversations that can support collaborative modeling behavior in science. (Characteristics of students' modeling practices are discussed in the next chapter). Then I turn to a discussion about what a productive conversation about modeling in science looked like in this study, and what were the characteristics of modeling conversations in the context of using CPEs for developing representations of physical phenomena. In the third section of this chapter I summarize prior discussions about students' activities and conversation in a different way, to talk about different approaches for using the two CPEs in the study. Finally, I include a discussion about the nature of different program strategies in MW and SC.

5.1. Students' activities and conversations while developing representations of physical phenomena

In this section I compare student behavior with the two CPEs with respect to (1) approaches to planning, (2) approaches to writing and debugging code, and (3) approaches to using the code of their programs as representations of the phenomenon.

5.1.1. Different approaches to planning

While planning their work with MW, *students talked about the structure of their programs* (see section 3.2.1). Like Joe and Samir who talked about having 3 sub-programs to represent the different parts of the phenomenon, and Aaron and Richard who talked about 2 sub-programs to represent jumping, students were breaking down the phenomenon under study into small pieces, based on the behavior(s) of the program's objects. Parts of the program in which an object had similar behavior were grouped together (e.g., the object moved in the same direction). Students avoided providing any details about the code of their programs, even though they seemed to have an idea of how their programs would be (see episode MW1). Samir for instance talked about that they wanted to have a routine for changing the angle of the moving arrow, but he did not talk about how much they wanted the angle to change. While planning their programs, Richard and Aaron started writing programs that included details such as amounts of forwards in their programs etc, but when they started talking about them, they talked about how their programs (that represent jumping on the moon and jumping on the earth) had combinations of forwards and waits with different amounts to result in different "speeds" of the jumping person. They also talked about the structure of their programs, but did not talk about the details of each program. In many cases they preferred to start typing code instead of talking about what they were thinking to type (see episode MW2). They also talked about the possible results of their program ideas (simulation) to support their program decisions. The simulation as an outcome of their proposed program was used to support their program ideas: a proposed subroutine would result in a particular simulation which would represent a particular part of the phenomenon under study.

Conversations about the structure of their program were starting points of productive conversations in science. MW students working with MW were breaking down phenomena into (programmable) pieces based on the identification of the parts of the phenomenon that shared similar objects' characteristics. In doing so, students had to start thinking about possible similarities and differences between the different parts of phenomena. The subsequent development of programs was based on the differences of the objects' behaviors (parts of the phenomenon with similar object behavior were represented by the same subroutine), even though students were not representing what was causing those changes in the objects' behaviors. For instance, Joe and Samir developed a program to represent an arrow flying in the air. Their program had two lines of code that corresponded to the two parts of the phenomenon that they identified: one while the arrow was moving upwards and one while the arrow was moving downwards. Each line of code was an independent subroutine that created a particular motion, without any relationship between the two subroutines. In a way, their program was “suggesting” that at some point of the arrow’s trajectory, the arrow switched from going upwards to going downwards. Students did not talk nor did they represent in their program what was actually causing the arrow’s behavior: one way could have been to have a particular subroutine that would make changes to the arrow’s direction. That is, rather than simply showing what happens in a sequence of events, students could have had code that could cause those events.

On the other hand, *students working with SC were planning their work by talking about the story (scenario) they were about to program*. Rather than talking about program ideas, students tended to focus on the overall story line and talked about what their simulation would look like (see section 4.1.1). Annie and Bryan for instance talked about their game being about shooting down helium balloons that travel at different speeds. They indicated that the player would use darts to shoot down the balloons, they talked about how scoring would be done and what the player was supposed to do, and after what score would she move to the next level. Zen and Seth, on the other hand talked about what was going to happen in a game about two boys running in a race, how one would run faster than the other, but have to stop more frequently to get some rest. The conversation was about a sequence of events that students would show through their simulation. Unlike students working with MW, students working with SC were providing a lot of details about their scenario, including details about the settings, the objects/characters in the story and how they looked, in addition to details about the scenario (for one example see episode SC1).

In talking about their scenario, students were breaking down ideas, in a number of sequential events: what would happen first, second, third and so on. It was almost like students were describing a movie, talking about each scene one by one. Simulation was used to justify their program decisions rather than used to support their program decisions, as in the case of students working with MW. Students needed to design a particular simulation, so they were talking about ways to develop programs that would cause that simulation. Thus, students working with SC talked about their simulation as a sequence of events that each one happens after another. Code was used to create this simulation.

Talking about representing scenes of a scenario is not a productive conversation for modeling in science. As previous research about the use of CPEs as modeling media suggests (Colella, Klopfer & Resnick, 2000), students working with SC were talking about the system (overall story line) they wanted to program and about the system characteristics and system changes. System behaviors and changes are caused by object behaviors, and more importantly modeling the system requires modeling objects' behaviors that would subsequently cause system behaviors, especially if you are using a CPE – like SC – which uses object oriented interfaces for programming. This means that the user cannot model a system behavior simply because a system is not an object, but consists of a number of objects. The programmer needs to think about the objects of the system and the objects' behavior that cause system behaviors and system changes. Students' stories were made up of a number of characters (objects) that had particular behaviors and interactions. Those object behaviors and interactions caused the system's (story's) characteristics and behaviors: talking about the system characteristics is easier then representing them, because they are caused by the behaviors of individual objects.

This seems to create a paradox: students were using an object oriented medium, which is based on the idea of directly programming characters with behaviors, but on the other hand, students were planning their work thinking about the system and not about the characters, possibly due to how a simulation looks in SC. A simulation in SC can easily be seen as a presentation of a story, e.g., of a race, even though what is going on in the race is based on how the runners act (one can run faster, but needs to stop frequently to get some rest). In this sense, talking about the system and the system changes was not a productive conversation in modeling, because students were then required to translate those ideas into rules about the system's object behaviors. Students working with MW, however, were planning their work by talking about their program's structure based on the objects' characteristics and not on the overall phenomenon.

These findings suggest that students were engaged in different "states of mind" (MW: talking about their program structure, SC: talking about an overall story line). Students working with MW were having an *authorship relationship* with MW: their work and conversations reflected an effort to write a program. Students in SC, however, were rather in a mode of *creating a visual simulation* with SC that would show their story.

It only makes sense that students would take these different approaches to planning, if you think about it, because they were using two radically different programming environments. Creating programs that run was an added difficulty for students working with MW, which students working with SC did not have because the act of programming in SC is much easier.

MW is a textual-based CPE, which requires particular program language for writing programs that create simulations (Papert, 1980). MW is an open-ended environment and does not provide any scaffolding for writing programs, which adds to the difficulty of having to write a program that can run successfully. This is also supported by the fact, that as soon as students got a program that runs (see discussion below) they shifted their focus on the simulation and its depiction. In sum, creating a simulation in MW has to go through the process of writing a program with no bugs.

On the other hand SC is a CPE that has a programming-by-demonstration interface, including a lot of scaffolding for rule creation (Kiper, et al, 1997). Scaffolding for programming may limit some programming capabilities but it makes the process of programming simple situations (such as uniform motion) very easy. Assigning behaviors to objects is as easy as turning the system into programming mode, in which SC records a desired behavior. In addition and as important, rules in SC cannot include bugs, but they can result in a non-desired behavior, and thus in most cases there is no need for debugging. Further, programming-by-demonstration enables students to "program" a cartoon-like simulation with no awareness that they are programming (Rader, Brad & Lewis, 1997), and thus students do not have to be concerned about the details of programming. Since students' purpose was to create a simulation, they were concerned with the details of their story line that would be represented in the simulation.

In the context of preparing to write a program to represent a phenomenon, MW seemed to be more supportive for ways that could trigger modeling conversations and practices than the way (visual relationship) that students had with SC. I do not suggest, however, that SC was a non-productive tool for this modeling in science. Rather, the way that students were inclined to use it early in their work was not supportive of ways that could trigger conversations, students' thinking and activities for modeling physical phenomena.

5.1.2. Differences in writing and debugging code

Students working with MW maintained an authorship relationship with MW during the early phases of programming. *Their goal was to type programs that would run* (see sections 3.3.2 & 3.2.3). Samir for instance, undertook the role of the "typist" and started typing their program whereas Joe was simply watching. Richard had a few brief conversations with Samir, only to indicate or talk about typos in the program and program primitives. When Samir asked me to see their program, several minutes later, their simulation looked ok (showing an arrow moving in the air) but their code was structured very differently from what they had talked about before.

At the beginning of their work, programming was the act of typing long lines of code which in addition to giving instructions about the object's behavior(s) had to include code for the program itself (e.g., a program in MW has to start with a "to <name_of_the_program> <variable's_name_if_any>, identify for which object was the program). Students in SC were simply assigning behaviors, mostly by demonstrating them to the system. Students' program in MW consisted mostly of a single subroutine, with a number of instructions of what the object would do sequentially. For instance, a program of a falling ball would look like forward 1 forward 2 forward 3 forward 4 forward 5 forward 6, and a program for jumping looked like forward 5 forward 4 forward 3 forward 2 forward 1, and then forward 1 forward 2 forward 3 forward 4 forward 5. These instructions seemed to have an underlying mechanism that was causing the change in the behavior of the object (in the above examples the speed of the falling ball and the jumping boy), but it was not represented in the program. Lastly, during typing, students' conversations were limited, and happened only in cases that the "typist" mistyped a primitive or was not sure about one.

Unlike typing, debugging was a process of going from the program window to the simulation window to run the program, and back to the program window to fix any bugs and so on (see section 3.3.2). Conversations during this time were also limited and students were focused on getting the simulation to run (see section 3.2.3). In this sense, the code was used by students as a tool, but in a rather non productive way at least for scientific thinking, since the goal was to get a simulation that runs (not a simulation that shows or represents a phenomenon or even as a communication device). However, it is important to note that to talk about how a simulation or a particular program represents a physical phenomenon (see discussion below about productive conversations for modeling), students need to have a program that runs and creates a simulation. It seems to be rather unlikely for students to have such a conversation if the program does not run, based on fact that as soon as they had a program that ran successfully, they started having a conversation about the simulation and/or the code as representation of the phenomenon. On the other hand, students working with SC did not have this added task, to get a program that runs. They had, however, the task of translating a story into rules for individual objects, which students working with MW were doing during planning.

The two problems, however, (1) type and debug code to get a program that runs in MW, and (2) translate a story line about what happens into individual rules in SC are different kinds of problems. The first is adding more work to students; before they can have a conversation about a simulation and its program they need to get the simulation to run. The second problem however, is more of a perspective issue: had students in the study seen planning as talking about the behaviors of individual objects, then they would not have any trouble during this phase of their work. Of course, there are possible ways that can make this part of students' work easier, such as providing students working with MW with some scaffolding for code writing. It is also possible that as more experienced with programming students get, they would move through this part of their work more quickly. The latter can possibly apply for students working with SC, too, whose experience with programming in SC may help them to think from the beginning in terms of the objects' behaviors. In addition, as I discuss in the following chapter, it is possible that there are phenomena in science that are based on the behavior of a single object (e.g., a ball falling), in which cases it is productive to think about the story for modeling the object's behavior, because the overall story line of the system being modeled is the story of the object.

When their program was bug-free, students working with MW started talking about the resulting simulation. They were concerned about how the simulation looked and as important whether it looked realistic (see section 3.3.3 and 3.2.4). *Their focus shifted to getting a simulation that looked "realistic"*. For instance, Richard disagreed with Aaron’s program, because his program was showing the boy on the Earth jumping much slower than the astronaut on the Moon. Richard was suggesting that their simulation should show the astronaut jumping slower than the boy on the Earth. Aaron on the other hand was thinking that because of the greater gravity on the Earth, their simulation should show the boy jumping slower (more waits in the earth program).

The MW students’ conversations during this time were about what possible changes could be made to refine their simulation. For this reason they were supporting their ideas using observations and experiences from their everyday life. Once again, the code was used as a tool, to modify their programs to result in better simulations.

The structure of their program was not a concern any more, but rather, MW students were making *any* necessary changes to improve their simulation. In several cases (e.g., Joe and Samir’s program, Nick and Tilson’s program), changes departed significantly from their plans, indicating that the students were focused entirely on how the simulation looked and not on how to represent the phenomenon they were studying.

When students working with SC moved from planning to start programming, they *talked about details of their scenario in an effort to translate them into programmable rules* (see section 4.1.2). For instance, just before starting to make rules, Annie and Bryan were talking about their game (e.g., have balloons that move in different speeds) and then talked about how to make that happen (create a rule for the red balloon to move quicker and a rule for the blue balloon to move slower). Unlike students working with MW (who were typing and debugging code to make their program run), programming with SC was an interactive process of work and conversations among students, probably due to the dynamic process of programming. Students did not have any clear plans about how exactly to create rules in their designs, nor did they work alone as students working with MW did. Rather, while working, SC students tried to translate all the details of their story into rules. *Their focus was on creating a simulation that would show their story*. During this part of their work, SC students wrote, deleted, and re-wrote rules in an effort to create a simulation of their story. The code is solely used to create a simulation that would demonstrate their story, as a succession of events, one following another. In this sense, students maintained the focus on an overall story line, which they had during planning, to tell a story as a succession of events.

Because of the students' focus, programs in SC were successions of rules representing "scenes" from their story, which SC was running in sequence. Also, unlike students working with MW, SC students wrote each rule and tried it immediately before moving to the next. If a rule did not have the expected results, students did not spend any time figuring out what was wrong. Rather, they deleted the rule and created a new one (see section 4.2.2).

There is not conclusive evidence about whether students deleted a rule only to recreate the same one or whether they specifically knew what to do differently in the new rule. In most of the cases students did not try to see what was wrong with the old rule (see episode SC5 in chapter 4). Early in their work (when most of the rule changes take place), when their programs consisted of only a few rules that were relatively simple (did not include a lot of behaviors or any variables) students seemed to know what was wrong with particular rules but still preferred to simply recreate them rather than edit them and make corrections.

This is not to suggest, however, that students were deleting their rules without being sure what was wrong with them, because there is some evidence of the contrary (see section 4.2.3). At least in some cases, students were talking about what was wrong with their programs (in some cases they specifically talked about particular rules) and simply instead of editing and making changes in their rules they deleted their rules and created new ones.

In addition, debugging was only in the form of deleting rules that had unwanted effects or changing the rule sequence. Syntactical bugs are almost impossible to be found in SC programs because the software provides a lot of scaffolding for creating new rules. The user has simply to fill in blanks about the conditions of each rule (when SC will run it) and the effect of the rule (desired behavior). In fact, in some cases, programming was simply done by demonstrating to the system the desired behavior.

Once again students using different CPEs in the study were seen to operate in different states of mind. Students working with MW seemed to have shifted their focus to having a visual relationship with MW, focusing on getting a simulation that would look realistic. On the other hand, the act of developing rules, caused students working with SC to shift into having an authorship relationship in an effort to write a program that would show their story, similar to what students working with MW had during writing and debugging their code.

5.1.3. Differences in using code as a representation of the phenomenon

There was a third shift in student focus during their work with CPEs that was mostly observed with students using MW. This was a shift to use CPEs as modeling media. For this shift to occur, students working with MW had to start reading their code, instead of simply running it to see the resulted simulation. In some cases this shift happened when a student was looking at another group’s code.

When students working with MW started reading and talking about the code in their programs, students referred to code as a representation of the phenomenon (e.g., representation of the behavior of the objects in the simulation), using mathematically precise code to talk about how the phenomenon occurred. They also used experiences they had from everyday observations, in order to support ideas about how the phenomenon occurs. Debates, such as whether jumping on the moon is slower vs. higher than jumping on the earth, or whether an arrow would travel in an oval shape or not were supported by experiences that students had from similar phenomena.

Conversations about the representation of the phenomena in MW code resulted in iterations of model refinements, mostly in an effort to include a mechanism of how the phenomenon happened. Rather than seeing code in the programs as a tool that creates a simulation, students started seeing them as a representation itself. Therefore, rather than seeing programs as simple descriptions of the phenomena, students started seeing them as causal models that caused the phenomenon. When Nick, for instance, saw Joe and Samir’s simulation about the moving arrow, he indicated that it looked fine. As soon as he saw the code that created the phenomenon, he indicated his puzzlement: “[…] actually this program isn’t what an arrow does! […] that’s what a rock does. What the program is doing that would what a rock does. This isn’t what an arrow does. An arrow drops just like a gun bullet does.”

Because students' focus was now on the code as a way of representing the phenomenon (simulation was another way that a phenomenon can be represented) students also started to talk about how the phenomenon happens. For Nick to identify that "this is not what an arrow does" required him to see in the code something different than what he expected to see in the case of an arrow moving in the air. Even though the simulation looked "fine" the code in the program did not. This possibly suggests that what Nick was seeing between the lines of code in the program was a causal mechanism, different from the one he expected. He was expecting to see one kind of causal mechanism to represent the phenomenon but instead the code was reading something else. That is why he suggested that the program was representing a moving rock rather than a moving arrow. In this sense, starting to read code (instead of simply running the program) helped students focus on the code as a representation of the phenomenon and also make them look for a casual mechanism of the phenomenon.

It is important to note, however, that this kind of conversation is still far from a conversation about the underlying mechanism of the phenomenon. Students working with MW at this point were developing descriptive models (for a detail discussion see next chapter), of what happens in the phenomenon. They were not developing models in which the code was a representation of the causal mechanism of the phenomenon. In the above example, the conversation led Nick to propose a different program that "better" represented the motion of the arrow. That program however, was still lines of code of successive events rather than a mathematical mechanism that would change the angle of the moving arrow while moving in the air. What had changed, however, from before was that students were now more consciously *looking for* the causal mechanism that was underlying the phenomenon in the code that represented the phenomenon. The focus on the code seemed to be more likely to trigger conversations about what was actually causing the phenomenon.

Talking about causal mechanisms, students working with SC had to shift their focus from showing a story through the simulation (that is assigning behaviors to objects in the story) to talking about concepts (variables) in the story that affected object behavior such as food energy or speed. Like students working with MW, students working with SC were having conversations about how things happen in their simulation, utilizing the scaffolding of programming of SC to talk about the causal mechanism of the phenomenon. Zen and Seth for instance, were using the number of Gatorade bottles that their runners can find and collect while running (they created a variable to account for the number of Gatorade bottles) to represent and talk about the "energy levels" of their runners. Later, they added rules for subtracting from their "Gatorade" variable depending on the runner's speed and slowly adding to the "Gatorade" variable when their runner was not moving.

Scaffolding that SC provides for rule creation seemed to support conversations about the mechanism; the difficulty however was to start such a conversation. Students seemed to see their work with SC as developing games (see discussion later) and focusing on the overall story, which seemed to get in the way of modeling, partly because to have a modeling conversation students had to talk about what was causing changes in the behaviors of the objects in their game scenario, as I indicate below.

To illustrate that difficulty of focusing the conversation on causality, I am presenting an episode of a conversation. This example is not included in the findings from SC chapter because it does not support a claim about programming strategies or types of conversation. Rather, I am using it below to illustrate a particular difficulty of using available tools to talk about a causal mechanism. This is an episode from a conversation I had with Annie and Bryan on the 2nd December. 2002. Annie and Bryan had started developing a microworld in which fishes were swimming around in a lake, with bigger fishes eating smaller fishes. The excerpt starts at a point where students wanted to create a mechanism that could account for and limit the number of fishes that bigger fishes eat. Part of this mechanism would enable each fish to start eating again, when the food was digested. Bryan was finding the solution he was thinking amusing. The fish at some point was going to "blast" out the bones of the fish it ate and thus it would be able to eat some more.

245.**Annie:** we want the limit [of the number of fishes it can it] to be 3. And then, after 3 it will stop for a few seconds and then, we don't know where it would come out, but then the, like bone figures would come down to the ground.
246.**Bryan:** it will blast and it will be bones in it.
[…]
236.**Loucas:** […]. Bryan what happens if you eat something?
237.**Bryan:** it, um ….it depends. Oat meal, rice…
238.**Loucas:** ok, Bryan let's, let's think, let's think you're hungry, and you eat something. What happens after you eat that?
239.**Bryan:** um, I am not hungry any more, <inaudible> a couple of stuff and then …
240.**Annie:** you have more energy!
241.**Bryan:** oh, yea, you have more energy, and if I have to I need to go… [to the bathroom].

When I initially asked what happens after you eat something, Bryan was suggesting that there is at least something coming out of your body when you go to the bathroom. As much amusing as this conversation was (!), Annie did bring up the idea of energy, suggesting that when you eat, your energy goes up and when you are hungry it gets low.

247. **Bryan:** when you're hungry you're…
248. **Annie:** your energy is lower.
249. **Loucas:** ok
250. **Annie:** because you don't have any <inaudible>
251. **Loucas:** ok, when you eat what happens to the energy?
252. **Annie:** um, the stuff from the food going to go to your blood stream and make your energy go up <inaudible>
253. **Loucas:** ok so your energy goes up?
254. **Annie:** yes, cause that <inaudible>
255. **Loucas:** how does your energy go down?
256. **Annie:** cause you, …
257. **Bryan:** you're hungry…
258. **Annie:** or the food that …
259. **Loucas:** how do you get hungry though?
260. **Annie:** the food, you have an upset stomach.
261. **Bryan:** um, how does it get empty?
262. **Loucas:** right, how does it get empty?
263. **Bryan:** yea seriously, how does it get empty?

As the conversation continued, Annie suggested that one feels hungry because she has an upset stomach, and when you eat the food gets in your stomach and then somehow energy gets into your blood stream, having possibly heard or read the latter. Bryan, however, indicated with puzzlement: "yea seriously, how does it [your stomach] get empty?"

As the conversation proceeded it turned to become about the mechanism of hunger and food digestion and I indicated that to Annie and Bryan. Annie then suggested the idea of nutrients in the blood stream, and Bryan indicated that you need those because you are getting bigger – the need for nutrients makes you feel hungry. Bryan continued to talk about that if something gets in your body (food) it needs to get out ("come out of the pores of your body" or "go to the bathroom"). He seemed to think in terms of a succession of events about hunger, almost like a succession of scenes that happen over time: first, one is hungry, then she eats, then food gets into her stomach, then it comes out of the stomach (through the body pores), and then she gets hungry again. Bryan was talking about the overall story line of getting hungry, giving facts about what seems to be happening. However he could neither talk nor be clear about what was causing the “scenes” in the story he was suggesting.

Even though the conversation was about hunger and despite their willingness to develop rules to account for eating limits and for enabling eating again, using SC to tell the story of hunger was evidently getting into their way of thinking about the mechanism that was causing the phenomenon. This was because representing events related to hunger could not represent what was causing the hunger in the first place. Further, there was not a clear connection between the successive events, other than their sequence. For instance, when they were modeling the motion of a falling ball, in each subsequent rule students increased the ball’s speed by one, using (without necessarily representing) a relation between the different rules. Representing that relation in their program was a step closer to representing the mechanism that was causing the phenomenon.

283. **Loucas:** how do they get hungry though?
284. **Bryan:** when there are no stuff in their stomach. They need because they’re growing.
285. **Annie:** cause they digested the rest of the nutrients and so ….
286. **Loucas:** so, if your stomach is full how does it get empty, […]?
[…]
290. **Annie:** because the nutrients in the food helps your blood stream.
291. **Loucas:** but why, would, what, if I’m not using, if I am not using the nutrients why, why do I need to eat?
292. **Annie:** to keep growing. To keep healthy.
293. **Bryan:** yea, so like there ….oooooou
294. **Annie:** it’s hard to explain.
295. **Bryan:** I know. But go like to McDonalds, he’s hungry so he’s ordering <inaudible> pizza, <inaudible> as he eats it goes down to your body and then it charges up the cells and then the stomach (I am not sure) pores so that he won’t have to starve, so that he won’t have to like starve any more. If you starve, you really, really need to have food again, you can’t like run for 2 miles if you starve. So it’s something about…

As the conversation proceeded, it started to become apparent that it was rather possible that Annie and Bryan did not have any experiences that they can use to talk about what exactly was going on in one's stomach. Even though Annie and Bryan were attempting to find an explanation for how that consumption occurred, they did not seem to have much experience with biological mechanisms. Bryan in line 295 started talking about a possible biological mechanism, but still he was trying to justify how one stops feeling hungry. However, the purpose of the conversation was to relate hunger (fishes cannot eat as many fish available) with energy consumption (at some point fishes can start eating again). I felt like they should have been able to talk about ideas related to energy storage and depletion. I decided to make an analogy to fuel consumption.

298. **Loucas:** Now think of this example. Let's say I have a car. Ok? And when the car, the car needs gas, right?
[…]
305. **Loucas:** so, if I put gas in my car, how does my car move?
306. **Bryan:** it'll like, there is like.
307. **Annie:** the gas goes into the engine …
308. **Loucas:** so the car uses up some of the gas to move.
309. **Bryan & Annie:** yea!
310. **Loucas:** so if I have at the beginning of my journey I have 100 gas, at the end of my journey I will have like I don't know 30, depending on the distance that I travel.
[…]
322. **Annie:** thanks. Food is like the fuel, and like the fish is like car. I mean it uses some of the food to keep going and then rest stays <inaudible>

The analogy of the gas consumption seemed to help them start thinking about food digestion. Food can be like gas, Annie indicated, which is consumed when the fish moves. Students stopped talking about events of hunger, and started talking about events that can happen to "energy". They were most likely in the same "state of mind" of telling a story, but this time their story was about energy, how it is regulated and how it is consumed. Moving consumes energy and eating adds to the energy levels.

Telling “the story of the energy”, a particular concept and a potential variable in this case, is very different from telling the story of a system that consists of multiple characters. The story of the system is a succession of events, which in order to be programmed one needs to identify objects’ behaviors that cause them and program those particular behaviors. Telling “the story of the energy” was using a “story-telling” approach in a productive way, putting together the pieces of the story about energy. This was productive, partly because different behaviors of “energy” (e.g., consumption, enrichment etc) can easily be different programmable pieces that can be represented in rules. In this sense, the story was (partly) the causal mechanism of a living fish, and thinking in that way was productive for developing representations of that mechanism. Had students continued to think in terms of a succession of events for the fish in the lake (eating, moving, etc) they might not have moved into productive modeling of the living fishes.

In part, it is possible to suggest that modeling, which involves representation of causal mechanisms, can start from telling a story of the “behaviors” of what is causing the phenomenon, whether that is gravity, change of speed, or energy. In one way, hunger in the above example can be explained by the behavior of the energy, the levels of which gets “higher” when nutritious food is consumed and “lower” when e.g., the person moves. In a different example, gravity may affect the angle of a moving arrow, or the vertical part of its motion without affecting its horizontal motion.

5.2. Productive conversations for modeling in science

In the context of developing representations of physical phenomena, students in the study had many conversations while working (or not) with CPEs. Some of their conversations, as previously described were more “productive” than others. In this section, I demonstrate how productive conversations look, within the context of developing models of physical (and other) phenomena.

Conversations as a data source for research related to educational settings have been the focus of linguistic approaches of analysis (Edwards & Mercer, 1995). Traditionally however, research in education has not used conversations among students as a data source, partly because these kinds of data can provide less information (than clinical interviews for instance) about students’ abilities for argumentation, abilities for coordination between theory and evidence (Kuhn, 1989), controlling variables etc. As I discuss in Methodology, investigating student abilities in science, such as those that I previously mentioned, has traditionally been the focus of research of educational psychology, which does not use classroom conversations as a data source. Indeed, classroom conversations are open-ended exchanges of ideas and arguments, where adults are hardly (or at least they should be) part of the conversation. Thus, students conversations are thought to be unstructured general conversations (something however that I will try to argue against).

Research in science education has not used widely analysis of students’ *theoretical conversations* (that is conversations about a phenomenon without any experimental data available), because a popular approach for teaching science has been coupled with “hands-on science.” That implies that successful approaches for teaching science should have students manipulating experimental apparatus, and carrying out experiments. Traditionally, the core of science learning has been the experimentation: children (like scientists) design experiments, collect and analyze data, and develop or modify ideas (theories) based on the experimental results. Research in science education has focused on students’ abilities to design, carry out experiments and interpret results.

However, student conversations are usually the core of the everyday learning: students spend most of their days in schools: when the teacher does not lecture, and students are not writing, they talk to each other. Even when students talk about results of their experiments, analysis of their conversations has much to say about their abilities for scientific inquiry (Louca, Hammer, & Bell, 2002). More importantly, students’ conversations can serve as sources for teacher diagnosis (e.g., Bell, to appear) for students’ actions and student thinking during the process of learning. I address this issue on the next chapter entitled “Steps towards modeling.”

In traditional views of science education, the approach that I have followed in this study was to engage students in theoretical science. Students did not have any experiments to perform and nor any results to interpret. They were simply asked to develop representations of physical phenomena. A common reaction to such approaches, following in part “The National Science Education Standards (NSES, 1990)”, is that students are doomed to fail. According to the NSES, in early grades, the emphases are on science as empirical inquiry, to help students develop abilities for observation, experimentation, forming conclusions based on evidence they obtain through experimentation, and on the logico-mathematical abilities for controlling variables and organizing data. Only at grades 9-12 do the standards emphasize theoretical inquiry in addition to empirical, such as in the expectation that students should "formulate and revise scientific explanations and models using logic and evidence" (NSES, 1990, p. 175).

However, as I discuss below, analysis of student conversations from this study indicates that there are instances where students' conversations are productive even without any empirical evidence available. Productive conversations in science can take several forms, e.g., focused on argumentation (Louca, Hammer, & Bell, 2002), and on abilities for coordination between theory and evidence (Kuhn, 1989). What do productive conversations for modeling in science in this study look like? That is, when developing models of physical phenomena, what do productive conversations look like?

In the discussion that follows, I talk about what kinds of conversations are supportive of collaborative modeling practices, in the sense that they can (or do) get students to talk about modeling physical phenomena.

5.2.1. Conversations about causal mechanisms in SC

Let's consider two different situations, taken from the same students from this study (students working with SC). In one situation, students were presenting their ideas to be programmed. They were focused on the story of a game that they decided to create and they provided details about that story. They decided to create a game in which different-colored balloons would fly in the air, some faster than others (for details and transcript see episode SC1). The player would shoot down the balloons using darts and she would gain points: faster balloons would be worth more points than slower balloons. Balloons that would not be shot down would stay on the top of the screen and a bird would fly around to pop them. They then would fall on the floor and a girl would walk around to pick them up.

This was a description of what would happen in a game. It was a description of a scenario to be programmed because students, following a narrative perspective, talked about things that would happen in the scenario of their game without any reference to how they would program them or what was causing different things to happen.

A few meetings later, the same students were talking about their balloons. They were also telling a story, but this time their story was more specific (for more details and transcript see episode SC2). Balloons, they indicated, would travel in different speeds based on the amount of helium inside them. The more the helium they had, the bigger the balloons would get. When a dart pops a balloon, the balloon would quickly start losing helium, to distinguish between losing helium when real world balloons have a hole and when real world balloons are intact (balloons with no holes would lose their helium in a few days).

In both cases, students were presenting or telling a story. In the first case the story was about their game, and in the second case the story was about a character/object in their game. The first story was a general story about what one would see in their game; it was the story of the simulation that they would design. The second story was a much more detailed one, about a single balloon. Students were talking about the different properties of the balloon (amount of helium), the relations between entities (helium and balloons) and different behaviors of balloons (rates of losing helium).

Unlike the first one, the second conversation was more supportive for modeling in science: this kind of conversation could make it more possible to trigger conversations about modeling parts of phenomena. Students were focused on a single object, telling a story about that object. In this way, they were breaking down the behavior and characteristics of the object (balloon) in small meaningful programmable pieces of knowledge, which can be easily translated into code for SC. The difference between the details in the first conversation from those in the second made the second conversation more likely to trigger modeling conversations: in the first story, different colored balloons would travel at different speeds while in the second story, balloons with more helium would travel faster. Details in the first conversation were about depicting details of their games, whereas details in the second conversation were details of the behavior of different objects.

5.2.2. Conversations about causal mechanism in MW

The essence of modeling is developing representations of phenomena that include or are the mechanism that causes the phenomenon (see introduction and discussion of next chapter). In this sense productive conversations for modeling physical phenomena are conversations about the mechanism that causes (and can explain) the phenomenon. How the difference of “gravity” on the moon and on the earth affects jumping of the same person, and how that can be represented in code, for example, was a conversation about a mechanism of what causes different “types” of jumping on different planets with different gravity (example taken from Aaron and Richard conversation, partly presented in episode MW4).

Conversations about causal mechanisms occurred specifically in the context of modifying programs. To move from a program that was simply showing different object states or object behaviors, to a different program that included a relation between these different behaviors or states, required students to think and talk about what was causing the phenomenon. Different speeds for example are caused by “gravity”. Describing different speeds is different from showing how different speeds are caused. Students usually started by writing programs that showed what the objects were doing first, second, third etc., usually applying ideas about what was causing the difference in that behavior, without representing them. This was usually a descriptive program, representing the different object states or behaviors in subsequent times.

One implication is that conversations that were supportive for modeling in science took place in a particular context, the context of moving from a descriptive model to a causal model. This is to stress that an interpretation about students’ abilities following a developmental perspective might be invalid: if students were not put in that particular context they would not be seen as having those abilities. This is also to support the importance of iterations during the process of developing models: if students are using a software that for instance makes it difficult to read and modify their code, this might not be an appropriate context for triggering modeling conversations. It is also important to stress that students were able to have such conversations. In fact, as I discuss later, in one group students had difficulty identifying the mechanism of the phenomenon, and they decided not to continue on working on that program! Not only were students able to have conversations about causality in physical phenomena, but they also had the ability to decide not to proceed, when they were not sure about what was causing the phenomenon.

According to NSES (1990) students in early ages should have conversations about forming hypotheses and later drawing conclusions based on evidence they obtain through experimentation. This suggests that students need to or at least should use empirical data to guide their interpretations of results and their conclusions about physical phenomena.

Without any empirical evidence readily available, when engaged in conversations about objects' behaviors and what was causing the phenomenon, students in the study turned to experiences that they had from their everyday life and the simulation of their programs. Usually conversations about physical mechanisms were conversations about detailed program decisions: should we have the person on the moon jumping lower or higher from a person jumping on Earth; should a dart travel for a short distance before starting to fall while moving, or should it start falling while moving from the beginning of its motion. In other words, students were debating different ways of representing behaviors of objects and in that sense, they had to support those ideas in ways that they could convince the others. Thus, they turned to whatever experiences they had.

Students were also able to distinguish between direct experience and logical conclusions. In the next chapter, I discuss the case of Annie and Bryan, in which Annie was suggesting that balloons with more helium can fly faster in the air. Bryan could not accept this, indicating that this was not necessarily true. Rather, he indicated, the more helium the balloon had the bigger it gets – this was a direct experience. But what the speed of balloons with different amounts of helium would be, at least for Bryan, was not a direct experience, and he couldn't "necessarily" agree with Annie – also indicating that her idea might have been plausible.

5.2.3. Code as a mechanism representation

Reading the code of their programs (rather than simply running the program and watching the simulation) seemed to be supportive of conversations about modeling physical phenomena. Reading the code, MW students had to follow instructions or rules for each objects in order to understand what the program was about. In this sense, code was a detailed representation of the phenomenon that students could see, talk about and refer to (rather than talking about ideas that they were thinking of programming). For instance, different ways of jumping caused by the difference of gravity on different planets had to be represented in particular ways, with some mathematical precision. Jumping higher vs. jumping slower had different code, and MW students could read and talk about that code. In other words, because of the act of reading code, students were putting themselves into a mode of talking and responding to each other about it.

5.3. States of minds: writing a program vs. writing a game

One of the differences in the way that students used MW and SC is the way students conceptualized their work with the programming environment. Students working with MW saw their work more like formal programming, whereas students working with SC saw their work as making games. This is an important distinction, because it was related with the ways students used available tools in MW and SC while developing representations of physical phenomena.

Generally speaking, students working with MW had more technical conversations. When presenting their program ideas, they talked about the structure of their program (e.g., Joe and Samir talking about their programs having different “parts”), the number of subroutines in their programs and how that would fit with a possible simulation as a result of the program, and about the code that they would use in their programs.

Students working with SC, on the other hand, saw their work as making 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 (e.g., during planning Annie and Bryan were talking about the details of their balloon game and Zen and Seth were talking about what was going to happen in a game about two boys running in a race). In doing so, students talked about the overall scenario of their game, following the order of what would happen first, second third and so on while their game continued.

Programs written by students working with SC usually consisted of a number of independent rules, one for each “scene” of their game. Students were breaking their programs based on the ideas they wanted to show in those scenes (e.g., a ball falling with a different rule for its different speeds) whereas students in MW were breaking their programs down in segments that were meaningful for what they represented: jumping consisted of two subroutines, one for the boy going up and another for the boy coming down.

During their work with SC, students’ program decisions were also based on features of their games. Students were not concerned with science as much as they were concerned about whether the behavior of different objects would fit in their game scenario. For instance, while Annie and Bryan were talking about programming darts to shoot balloons down, Bryan suggested that if they had the darts to drop while moving (an idea suggested by Annie) then darts would not be able to shoot down balloons on the other side of the screen (because it was too far for the darts to travel). In this sense, there were cases in which “game” and “science” were in conflict.

5.4. Assigning behaviors vs. giving instructions

One of the differences between MW and SC is the way programming is done. In this section I discuss implications from the study's findings about how students used the different ways of programming and what effects they had on student modeling of physical phenomena.

Programming in MW is done by writing instructions for turtles to follow (Papert, 1980). Students have to utilize program primitives in order to develop programs. This programming process makes MW an open-ended CPE, where students are not given any scaffolding while programming. Programs in MW have, however, to start and end in particular ways, that are not important for the program as a model representation, but are important for the program to run successfully.

On the other hand, programming in SC is done by assigning behaviors to objects in the form of rules (Cypher & Smith, 1999). Each rule usually has to create one behavior. SC provides some scaffolding for programming, presenting to students screens where the system records the desired behavior that is exhibited by the student or where students can manually assign the conditions and the actions for each rule (Smith, Cypher & Tesler, 2000). In addition, and in contrast with MW, SC also provides some scaffolding for adding and using variables and object characteristics, which students could use for programming ideas such as "energy points", number of fishes eaten, speed, vertical and horizontal position etc. Programs in MW can also include variables, but students have to include them in the code of their programs, without any special separation of turtle instructions from variables.

Due to the different ways of programming in MW and SC, several differences were detected about the models that students developed and about the ways that students used programming. Surprisingly, there are a few similarities, too.

Early programs. Students working with SC, usually started their programs with a number of rules that assigned different behaviors to objects in subsequent times. In Zen and Sean's program about a dropping ball that was speeding up (see detailed discussion in the next chapter), each subsequent rule had the ball to move a longer distance. In this sense, programs were collections of different rules, that were not (and could not be) related between them. Rules in SC cannot have relationships among them, other than having SC to run a number of rules one after another. Thus, accelerated motion was represented by a number of rules that had different ball speed in different parts of its motion. The program was simply a collection of snapshots in which the ball had a different speed.

Students' programs in MW included a number of instructions that resulted, for instance, in accelerated motion. Instructions however did not include the mechanism that was causing the change in the speed of the object. For instance, a program for jumping looked like forward 5 forward 4 forward 3 forward 2 forward 1, and then backward 1 backward 2 backward 3 backward 4 backward 5. Jumping begins with a larger motion at the beginning, slows down and then accelerates again while the person jumping comes back to the ground. Despite of the fact that these instructions seemed to have an underlying mechanism that was causing the change in the behavior of the object (in the above example the speed the jumping boy, and how it changes), the mechanism was not represented in the program. In a way, students started by creating simple descriptive models for the phenomena under study. The difference with SC was that programs in MW usually consisted of one sub-routine causing different events in a sequence, whereas in SC programs consisted of multiple rules.

As I indicated before, students working with SC started planning by focusing on the overall story that they would program, whereas students working with MW were breaking down the phenomena in small pieces that shared similar, most of the times visual characteristics. However, it is possible that it was easier for students working with SC to make the move towards a single rule that would include a mechanism which would cause a change for example in the speed of the ball, because their previous act of writing rules possibly made them think in a way to break the motion of the ball into small pieces: the first rule has the initial speed of the ball, the second rule has an increased speed of the ball and so on. To proceed with a program that would include a variable that would reflect such idea, programming in MW was rather harder for students, possibly because of the lack of any scaffolding from the environment.

In a way, planning for programming and programming with CPEs seemed to be two very different processes. Unlike what one would expect, to see students plan in the ways that they can later use to program, students' actions during programming were departing from what they planned doing or at least of what they talked during planning. This suggests a possible implication about the mental models that students developed about the phenomena they were representing. Because of the way programming is done in SC, students may be more likely to develop mental models of a series of semi-independent parts of the phenomenon, whereas students working with MW seemed to create only one subroutine because it was easier and faster. They were also grouping together (e.g., on the same line) instructions about their different parts of the phenomenon, but this was just for visual purposes: in those parts the ball was moving upwards, or the arrow was moving horizontally.

On the other hand, programs in SC that students created were less advanced than the programs written by students working with MW. This was possibly caused by the fact that SC makes simple programming easier for students, but complex programming harder, sacrificing at the same time advanced programming capabilities for the sake of ease for programming. When students started thinking for ways to include causal relationship between snapshots of their program, they usually started to think about having 1 or 2 rules that would create such behaviors (and include a causal mechanism that would modify that behavior). At the same time, they would shortly find out that a single rule cannot include many different behaviors or a complicated causal mechanism, whereas there was not such a limitation with programming in MW.

There was another major difference in the programming with the two CPEs in the study that affected the ways students used them. Program representation in SC is done by graphical representations of the rules. These representations are simply reminders of what the rule does, because it is difficult to represent the dynamic process of the simulation into a static graphical representation. This representation was designed to avoid adding the difficulty of learning a program language to read and write program. On the other hand, programs in MW are written in one dimensional written program language, that students need to read in order to understand what the program is doing.

This seems to create a paradox: the process of programming in MW is more formal and thus possibly more difficult, because students have to manipulate a symbolic language in order to write instructions for the objects in their microworlds, whereas programming in SC is much easier and tangible because students are assigning behaviors to characters by simply demonstrating the desired behavior to the system. On the other hand, program representation in MW seemed to be much more meaningful, because students could simply read the code to understand the program, whereas program representation in SC was more abstract. Students in SC had to figure out what each rule was doing, by interpreting graphical reminders of the actions of each rule. Thus, reading their code in MW was much more often observed and it seemed to be related with different tasks by the students.

Debugging in SC was usually done by deleting a rule and creating another one in its place. Most of the times students did not try to identify what was wrong with one of their rules and then try to revise it. Of course there were some cases were students at some point were reading their programs in trying to fix the science they represented. Debugging in SC was not done in an effort to get a program to run, because programs in SC cannot have syntactical errors.

Debugging in MW had several different types. Students would debug their code to make their program run, they would change their code to correct depicting details in their program and would change their code in order to change the science represented by their program. The first two types of debugging were not related with modeling: in the first type students were trying to fix syntactical errors in the primitives they used and in the second type, students were focused in changing depicting details of their simulation, to make it look better or more “realistic”.

The third type of debugging, had been a source of productive conversations in science, where students were using the written code as the basis of their conversations. A ball can accelerate by adding 1 to its speed or by doubling its speed: these are two different ways of representing the cause of the speed change, and students had to talk about them to decide on which of the two ways to proceed.

Conversations about programming were also different. Prior to any programming, students working with MW would talk about their program structure and program details e.g., talk about breaking a motion into several meaningful pieces and talk about how to program those pieces. While programming, however, conversations were limited and usually programming was carried out by one student. After students had a program that would successfully run, they would have conversations about the science that was represented in their program.

Conversations among students using SC were less technical. Prior to any programming, students would talk about the scenario of their game or phenomenon they would program, without, however, providing any details about how they would program them. During programming, students had extended conversations in an effort to translate the scenario into programmable pieces and rules. Still however, their conversations were not about the mechanism of the phenomenon.

Interpretation of findings, however, cannot be conclusive about students’ work being more useful with one CPE over the other. Rather, I am suggesting that features of the one software seem to be useful in one context and features of the other software seem to be useful elsewhere. For instance, having conversations about the structure of a program, might sound technical and with little relation with science. It might however be more related to modeling than talking about different “scenes” of the phenomenon. Rather than breaking programs to pieces that were as small as “snapshots”, students in MW were breaking down their programs in ways more related to the phenomenon. For instance, to have 3 programs, each for a different part of the motion of an arrow (one while the arrow moves upwards, on while the arrow moves horizontally and one while the arrow moves downwards), is a representation of how students think of representing the particular phenomenon. Their program structure was related to different phases of the phenomenon they were representing.

Subprograms as small as subsequent “scenes” of the phenomenon were also a possibly meaningful representation for developing a model for the phenomenon. This is actually one of the first steps in modeling of biological phenomena. Phenomena in biology usually consist of a series of events that happen one after the other in sequence. To make sense of the mechanism that underlies the phenomenon, one needs to see all of these events and try to summarize them in a way that would result in a causal relationship among them. Using a number of subsequent scenes, students could easily craft a relationship among them and then create a program that represented that relationship causing the different scenes. In a sense, it was going from a descriptive model to a causal model.

6. STEPS TOWARDS MODELING

The purpose of this study was to investigate how students use CPEs for developing models in science. This study was based on the idea that the program language can become a design medium for developing models of (natural) phenomena. Students can use the tools that different CPEs provide to develop scientific models that go beyond simple description of the phenomenon, to include representations of the mechanism that causes the phenomenon.

In preparing for this study, the initial plan was to have students working with both CPEs to develop models of physical phenomena, mostly related with kinematics. During the first phase of the study however, some of the students seemed to lose interest while working with particular tasks with computers, whereas in other cases they were very enthusiastic about what they were doing. Thus, at the beginning of the second phase of the study, I decided to frame their work in an open-ended way: I decided to let them work on making representations of whatever natural phenomenon they wanted, in an effort to get them excited about their work with CPEs. As I describe in methodology, students were left to decide what they wanted to work with, with the requirement that their designs should include something from science (e.g., a phenomenon). Because of that, some of the groups chose to model a phenomenon that did not belong to physical science. Below, table 6.1 provides a summary of students' designs during the second phase of the study.

SC groups	Designs	Natural phenomenon
Annie & Bryan (1)	Game: “Balloon shoot-out”[6]	- Balloons traveling with helium - Darts to shoot balloons
Annie & Bryan (2)	Game: “Fish pond”	Regulation of eating & digestion
Seth & Zen	Game: “Race”	Energy consumption in humans while moving
Tyra & Sean	Raining	Rain cycle
Sean & Zen	A ball falling of a cliff[7]	Accelerated motion
MW groups		
Joe & Samir	Jumping on the Moon vs. jumping on the Earth	Accelerated motion
Aaron & Richard	An Archer	Projectile motion
Nick & Tilson & LJ	A boy on a train	Relative motion
Jiana & Gabriella	Meteors	Collisions

Table 6.1. Summary of students’ designs during the second phase of the study

In previous chapters (chapter 3 & 4) I have provided descriptions of how students use CPEs and in what particular context. In addition, I have provided some discussion about what productive conversations for modeling in science looked like in the context of developing models in science with CPEs in this study. In this chapter, I talk about the idea of using programming as modeling, highlighting instances of student’s activities and conversations, during which students were engaged in modeling practices.

Although programming media can be used as modeling media, students in this study were not always using them as such. In fact, as previously discussed in the differences in students’ approaches to planning, writing and debugging code and using code as a form of phenomenon representation (chapter 5, section 5.1), students used CPEs in a variety of ways: as depicting tools, simulation tools, programming tools or modeling tools. This is also a known theme in science education research community that investigates modeling (diSessa, et al, 1991; Louca & Constantinou, 2002; Penner, et al,

[6] During the third meeting of the second phase of the study, Annie & Bryan decided to abandon their balloon game and start working on game with fishes in a pond.
[7] Students worked on this during phase I. Reference to this episode is meant to illustrate modeling practices.

1997). Students can be seen as having abilities to use modeling media, including computer-based programming environments in a variety of ways that may or may not include using them as modeling tools. In part, the problem may be that children are usually familiar using the modeling tools in other settings, e.g., paper-and pencil tools (diSessa, et al, 1991; Louca & Constantinou, 2002), 3-D structures (Penner, et al, 1997), which may include physical experimental settings (Louca & Constantinou, 2002) and computer-based programming environments are some examples of media that can be used as modeling tools. Students may be accustomed to use the particular media for purposes other than modeling – usually as simple visualization tools – and thus they need to perceive these media as modeling tools before they will use them as such.

In this chapter, I start by describing the three emerging frames that describe students work with CPEs in this study: the programming frame, the visualization frame and the modeling frame. Then, I provide some discussions about shifts of students work and focus from one frame to another, following and analyzing two conversations that happen with students working with CPEs in the study: one conversation with MW and another with SC. Then I present three model episodes of student conversations. The first involves possible difficulties for modeling dynamics, the second involves distinguishing between programming and modeling and the third provides data from a conversation in which students treat code as a representations of the mechanism that causes the phenomenon.

6.1. Frames that describe student activities and conversation

Throughout this study, I have identified three different ways that students used CPEs. This identification was based on the different activities and conversation types that were revealed by analysis, and that can describe students' work with CPEs in the study. To describe these different ways, I introduce the term *frame*, to describe student activity and conversation focus during the study. By frame, I specifically mean a mode of reasoning with its own goals, strategies (including ways of using CPEs), and criteria for success. Students in the study were in a situation in which they had to use CPEs to make representations of physical (and in some cases biological) phenomena. In this context, the study's findings suggest that students were using MW and SC in several ways, possibly caused by their goals in each part of their work. These are the programming frame, the visualization frame and the modeling frame.

6.1.1. The programming frame

In the *programming frame* students were focused on the code itself. While working through this frame, MW students were focused on getting a bug-free program (their goal). Their work was not related with the actual phenomenon representation because they were concerned about how to program a specific idea, talking mostly about what code to write.

To write a bug-free program, especially in MW's open-ended programming environment, students were turning all of their focus on the program language and how to use it to create a program that runs. Students were concerned about the details of the code in each of their programs, and their conversations were about those details and in particular about what program primitives to use in each part of their program (their strategies). Despite their plans about their program's structure and their program plans, and despite any issues that were previously discussed about the simulation that their program would create, their focus was only on typing up a program that can run (criterion for success).

An example of students working within the programming frame can be seen in episode MW5 in chapter 3, which presents a conversation between Aaron and Richard during programming with MW. Their conversation was technical, mostly about the code that Aaron was writing, typos he made, and possible bugs. In the episode Aaron quickly typed their first program and tested it. He went from the program window to the simulation window several times, trying to identify the bugs that were preventing their program from running. Changes during that time were only meant to make the program to run, without paying any attention to the structure of their program.

6.1.2. The visualization frame

In the *visualization frame* students used CPEs to develop simulations that show physical (and biological) phenomena. Their goal was to use the available tools (such as program language, debugging tools etc) to create a picture, or a number of successive pictures (like in cartoons) that were produced by the program code. Students were not concerned about the structure of their code, but only about the simulation and how it represented the phenomenon and its visual details (criteria for success), even though they might not be related to the phenomenon (e.g., include a person that would drop a ball off a cliff, in a simulation that seeks to show how a falling ball moves). Students were using the code/rules as a tool to create a simulation (strategies).

An example of students' work within the visualization frame can be seen in episode SC2 in chapter 4, which includes a conversation from Annie and Bryan's group.

In the conversation, students' focus was on the simulation and how that looked. During the conversation, students talked about their simulation and things to add to it, focusing on details that had to do with how their game looked. They also talked about adding new characters and how those would fit with the rest of the underlying story of their game, indicating once more that their focus was on using SC to show what they had in mind (e.g., a story or a game) rather than model physical phenomena that they included in the game.

Students were very precise about details of their story, e.g., what would happen to the balloons going up and then coming down, how to avoid them piling up on the top or on the bottom of the screen. Despite that, students did not refer to any details of how they were going to make rules that would result in a story like the one they were discussing. Their focus was to have a complete story, with all the characters and details in place, and not on the mechanism that causes the different speeds in each balloon. In addition, they were interested in showing *what* was actually happening in their design and not necessarily in showing *how* their rules created particular behaviors (e.g., blue balloon moved slow) and system characteristics (e.g., a combination of three balloons moving with different speeds created a particular pattern on the screen) . If depiction criteria were met, there was no further need for any other kind of rule or any conversation about existing rules.

6.1.3. The modeling frame

In the *modeling frame* students were focused on the development of programs that are representations of the mechanism that cause and can explain physical phenomena. In this sense, students seem to use the code and its structure are a representation of the phenomenon itself, rather than simply causing a simulation or being descriptive of what happens in the phenomenon.

The difference between the visualization and modeling frame is rather important: showing something that looks like the phenomenon, for example, a number of successive scenes in which the ball had different speed, can be described as work within a visualization frame whereas showing something that models the mechanism of the phenomenon, for example, a rule that changes the speed of a falling ball, can be described as work within a modeling frame. This distinction between the two frames has also been identified by Penner et al (1997) who asked students to design 3-D models that would represent the function of a human elbow. In the beginning of the study, students were focused on how to design their 3-D models to *look* like the real human elbow (focusing on depicting details such as skin color etc) rather than representing/modeling the *function* of the human elbow (focusing on the role of each part of the elbow for its function), which students did later in that study.

An example of a beginning of students' work within the modeling frame is represented in episode MW7, where Nick and Joe and later Samir had a conversation about how the code of a program was not representing the actual phenomenon (an arrow moving in the air) for which it created a simulation. In the beginning of that episode Nick was working within a visualization frame and had no problem with how the simulation looked. When later he had a look at the code that created the simulation, he indicated that it was the code for representing the motion of a rock that was thrown rather than an arrow moving that the simulation was showing. From a modeling frame, the program was not ok. Later, when Samir joined the group, students have a conversation about two programs that represented the same phenomenon differently and Samir started talking about the relation between the horizontal and the vertical motion in a projectile motion (Samir's contribution to the conversation is not discussed in MW7 and is presented in this chapter).

6.1.4. Possible implications of multiple frames

That students in the study were seen to work within different frames, having different goals for their work and different working strategies has several implications for using CPEs as modeling media in science education. Implications that I discuss below are related to teaching practices, to views about student abilities to use programming media in a variety of ways, and to methodological issues.

Due to a number of possible factors such as the short duration of the study, modeling instances (students' work within the modeling frame) were limited. However, when looking closely at the data, one is able to find brief instances when students attempted to use programming in ways that can be characterized as attempts to model physical phenomena. This, however, should not be interpreted as a failure of the CPEs to be perceived as modeling media. Throughout this chapter I highlight the use of CPEs as modeling tools, which seems to be associated with ways of thinking in science that are useful for focusing on the mechanism that causes phenomena in science.

Related and as important, the distinction between using programming for scientific visualization (visualization frame) and using programming for modeling (modeling frame) might be hard to detect. This distinction specifies when does programming stop being about writing programs that run, and starts being about writing programs that are representations of the phenomenon's causal mechanism(s). Being hard to distinguish can also suggest that it is possible that the two might blend into one another, and students may be seen working within one frame, but in some cases functioning within the other frame. In this chapter I seek to provide some descriptions of this distinction, in the form of detailed descriptions of modeling, and in the form of descriptions of shifts in students' thinking and work towards modeling practices.

If we want to promote student modeling, it is important to be able to notice the beginnings of modeling. Because they are hard to be detected, teachers need to be careful. They need to know how modeling conversations look and how the particular use of CPEs as modeling media look in order to be able to make the right diagnosis and promote productive student thinking for modeling. Partly, this chapter also seeks to provide and describe these "modeling instances" as examples of activities and conversations within a "modeling frame". In addition, following analysis of student conversations, I seek to provide some analysis of those "modeling instances", discussing their features and their importance for modeling in science.

A fourth implication for this chapter is a methodological one: due to the small number and brief duration of modeling instances that I present and analyze, the purpose of the chapter is not to describe any general emerging themes about modeling with CPEs. That is, I do not seek to make generalized claims about the "modeling work" of students in this study. Rather, it is a presentation of "modeling instances," in a way that they might be considered as possible contributions to a descriptive theory of modeling in science with the support of programming media.

A large body of literature and research has highlighted the importance of using modeling for learning about natural phenomena (e.g., Penner, 2001; Penner, Lehrer, & Schauble, 1998; Schecker, 1993; Constantinou,1996), and using CPEs as modeling tools in science, to provide learners with the appropriate tools to develop models of natural phenomena (diSessa, Abelson, & Ploger, 1991; Redish & Wilson, 1993; Sherin, 1996; Sherin, diSessa, & Hammer, 1993; White & Frederiksen, 1998; Wilensky & Resnick, 1999) and thus help learners develop some understanding about the natural phenomena in addition to learning about the modeling process it self. These studies have in most cases illustrated the importance of using CPEs in science through studies with students (e.g., Louca & Constantinou, 2002; diSessa, et al, 1991; Penner, et al, 1997). Another body of literature has discussed the implications of using programming media – mostly Logo-like media – for developing and transferring abilities (e.g., abilities for problem solving) to other domains of learning (e.g., Orhun, 1993; De Corte et al, 1991; Enkenberg, 1989; Verschsffel et al, 1989; Fay & Mayer, 1987). A third body of literature has documented the characteristics of different CPEs that have been developed to support young learners' needs (e.g., Smith, Cypher & Telser, 2000; Cypher & Smith, 1995; Rader, Brand & Lewis, 1997).

The above three bodies of research and literature have been using a variety of approaches to research but mostly have been conducted through particular research communities. The first through the research community of *science education*, the second mostly through the community of *educational psychology* (cognitive effects) and the third mostly represents a part of *educational technology* research community. Prior research however, has failed to talk about issues that are related with the synergy of the research in the above communities and are related with the particular ways that students use different CPEs in the context of modeling natural phenomena: (1) the increasing interest in developing computer programming environments for young children, motivated on general principle by the increasing role of computers in society; (2) the interest in children's science education, including helping children to come to understand the nature of scientific understanding and in particular the role of models in science. This has been a goal of this study, to investigate and document activity and conversation patterns of fifth grade students while using CPEs in science and describe how modeling can occur in this particular context.

Lastly, I would like to refer to a point made in a previous chapter about nascent student abilities for theoretical inquiry in science. The fact that there are many different ways that students can use CPEs in the context of developing programs/models representing physical phenomena, has a possible “psychological” implication: students seem to have abilities of different modes of engagement and thinking as well as abilities to make shifts between those modes of thinking and engagements. In addition, in the modeling episodes that I provide and analyze below, students were seen to make shifts in their activity and conversation focus, that was not facilitated by myself. Students started to operate within a modeling frame, possibly suggesting that they had the (nascent) abilities of thinking and working within different types of frames including a “modeling frame”. More importantly, one can hypothesize that students have abilities to detect different needs during their work and make shifts in their activities and conversations based on these needs.

6.2. Shifts between different "frames"

In section 6.1 I have talked about the different frames that students in the study have been working through, while using MW and SC to develop representations/models of natural phenomena. Because of the different ways that students used CPEs in the study, there were instances that students made shifts in the ways they used CPEs. Changing, for instance, "mode of work" from using MW as a visualization tool to using it as a modeling tool is important to highlight for three major reasons. First, that students can make those shifts based on the micro-context of their work (e.g., situation needs, their goals, the way they interpreted a task or a question), suggests that it is possible to argue that students have abilities not only to work within different frames, but also for changing their own work mode. Second, and related, it is important to study those instances when shifts happen, because they contribute to a body of literature and research that suggests that seeing students having or not abilities based on their "developmental stage" may underestimate what students can do (Metz, 1995). Students have abilities to work within different "frames" and with different modes of work, and studying them in only one context may suggest that students lack abilities that they otherwise can use (Louca, Hammer & Bell, 2002). Lastly, a third implication is for modeling in science. For modeling to take place, and given the difficulties that students may encounter for using CPEs as modeling media (see introduction for a justification), it is important that educators detect moments that shifts towards modeling, in order to promote them and support students thinking and student work towards modeling.

6.2.1. MW: Moving from the visual to modeling frame: from a simulation to the code

This discussion is about a group conversation between Joe and Nick and later Samir (for details and transcript see episode MW7 in chapter 3). The conversation was about Joe and Samir's program of an arrow traveling in the air to hit a target. During the conversation, I highlight two instances in which at least one student started to think in a modeling frame, with a focus on what seemed to cause the shift in students' thinking. In one Nick was seeing code as a representation of the phenomenon and in the other, after comparing two different programs, Samir was suggesting a possible relation between the horizontal and the vertical motion of the arrow. In the discussion that follows, I do not seek to claim that all students started to work within a modeling framework, but rather, to highlight beginnings of possible conversations and work within the modeling frame.

I started the conversation by asking about the code of the program, specifically pointing out to the students that I was not sure about whether their code represented the ideas that they had previously considered. The students, however, responded in a familiar way: they answered my question regarding their code by talking about how the simulation looked.

To help start a conversation about the code, I asked Joe and Nick to write a new program to represent throwing a rock, hoping that writing a slightly different program would provide a context for a discussion about code as representation of the phenomenon. The students decided to use a copy of the arrow program and modify it to match the new phenomenon. Even though this decision was unexpected, it had a very important implication. Instead of putting themselves in a situation where they had to write a new program from scratch, these students put themselves in a situation where they simply had to modify existing code.

Reading the actual code of the program, Nick, started talking about how the program was not representing the motion of a moving arrow. Nick, and partly Joe, talked about what in the code seemed to correspond to different situations and they started talking about what to change to make the code be more realistic. They subsequently modified the code to include Nick's idea about the motion of the arrow. Even though, conversational data do not support a firm claim about both students changing focus and see code as the representation of the phenomenon, this was a possible beginning of a modeling conversation and subsequent work in developing a model that would represent the phenomenon.

When Samir joint the group later, and after Joe talked about the two different simulations caused by the two programs, Samir started to talk about the dependency of the speed of the arrow to its motion. Due to the two different programs available, Samir started to think towards a direction of studying projectile motion, in which he seemed to suggest that how much the arrow would drop during its motion depended on the speed and the distance traveled in the horizontal direction.

<u>Part 1: Focused on visualization</u>

In the beginning of the conversation, students were responding to my question about their code, within a visualization frame, talking about how their simulation looked. There are two possibilities that this could have happened: the first is that students did not understand, or simply misunderstood my question to ask about their simulation. The second is that they understood my question, but they choose to respond within a different frame, with a statement that had a different focus from that of my question.

The distinction between the two possibilities is important for its implications for both instructional approaches and for student thinking. Understanding what the question was asking about, requires the ability to *understand the focus of the question*. I was asking about their code, but the students were responding by indicating their puzzlement: the simulation looked ok, why was I asking about it?

Another possibility is the students' *failure to detect the differences* between talking about code, and talking about the simulation. Students working within a visualization frame were focused on creating simulations that "looked ok," and represented the phenomenon they were intending. The code was simply used as a tool for developing the simulation, and students were focused on fixing depicting details of their simulation. One possibility is that working within a visualization frame, students could not detect the difference between a question asking about the code and a question asking how the simulation looked. They understood my question within the visualization frame.

As I have presented earlier, one of the emerging themes for students' thinking from this study, is that students may have nascent abilities for thinking in different ways. This, in conjunction with the data from episode MW7 may suggest that students were able to detect the difference between talking about code and talking about the simulation. Nick in line 10 suggested that "*but* that [the simulation] looks ok" indicating that there was no purpose to talk about the code of a simulation that looked ok.

247. **Loucas:** here is what I see. Repeat 30 fd powers and then left 0.5. so… if it is left 0.5 then you start like that, and then you go like that, little bit, little bit like that. So you go left, left, left and then repeat 40 right. So then it's right, something like that.
248. **Joe:** yea, that's right, that's the, that's what it does.
[…]
9. **Loucas:** ok. But if you look closely on your program, the program says that right? Because it's going left, left, left, and at some point it's going right, right, right. So if it's left it's going like that, and then … I understand that it looks like that, but the program actually does this, right?
10. **Nick:** but that looks ok!
11. **Loucas:** I understand that, but there are two different things: how it looks and what the program tells right?
12. **Nick:** but, well, what are, our purpose now is to show someone science, only, we are not showing then science by them going on this, to this thing (the code window) and…

As it can be seen from the conversation later on, students could work from different frames that have different foci, and responding to my question required a response within a different frame. Rather than seeing students to lack abilities for the different kinds of conversation in which I wanted them to be engaged, it is possible to suggest that we can see students being able to activate abilities to work and respond within different frames. In this sense, my question posed while students were in the modeling frame would be enough to have them start talking about their code as representing the phenomenon.

<u>Part 2: Nick's focus shifted towards the code as a representation of the phenomenon</u>

Following my prompt about writing a second program, students' decision to copy and modify the existing program (rather than working from scratch on a new program) put students in a different mode: they now had to read the code closely, trying to identify what and how to change. This led to a change in Nick's focus, from being on the simulation to be about the code. Nick, who had indicated earlier that their simulation looked ok, now indicated that the first part of their program (that resulted the upward motion of the arrow in the air) was not what it should had been. He suggested that things (like the arrow) that are shot horizontally continue to move straight and “drop a little” (line 31).

31. **Nick:** ok, that’s [the code] what a rock does. What the program is doing that would what a rock does. This is what an arrow does. An arrow drops just like a gun bullet does! A gun, like when you shoot a gun, the bullet would drop.

Writing and modifying code seemed to be associated with different *modes of engagement*: writing a new program in MW is a process usually taken care only by one student, group conversations were limited and students’ focus was on writing a program that runs (and not one that creates a simulation or represents the mechanism that causes the phenomenon). Modifying a program to fit science was associated with another mode of engagement: conversations were about the mechanism that was causing the phenomenon, and students supported their ideas for programming with experiences and observations of physical phenomena from their everyday life.

In the new mode of conversation, Nick followed partly by Joe’s contribution started talking about parts of their program, but the focus was on the function of the arrow and how it was represented by the program’s code. Reading the code of the program, possibly helped Nick to talk about the behavior of different objects in the simulation, and supported that by experiences that he had from everyday life. They talked about how the actual motion should look and what the arrow would do during its motion.

At least in one way, the simulation was not a context that would support a modeling conversation about the mechanism of the phenomenon, possibly because a simulation simply showed how the phenomenon looked. The code of the simulation, however, seemed to be a context in which a conversation about the mechanism of the phenomenon could take place. Of course, there was another important contribution, that of the “incorrect” code of the program which seemed to spark a conversation about two different versions of the motion of the arrow: the one that was represented by Joe and Samir and the one that Nick was suggesting, similar to the motion of a gun bullet.

Due to the disagreement, students resorted to supporting their ideas using experiences from the physical world, and inevitably the conversation was about details of the motion of different objects. Nick made an analogy to the motion of a gun bullet in an effort to distinguish it from the motion of a thrown rock. He was arguing that when shot in a horizontal direction, a gun bullet and an arrow keep moving straight and might fall a little; but they would not start going upwards as Joe and Samir’s program indicated. Of course if the arrow was aimed upwards then it would probably follow the direction that Joe and Samir’s program was suggesting. However, the code in their program had the arrow to start moving in horizontal direction, which was in conflict with arrow’s movement in the rest of their program (see figure 6.1 for details of their code).

Why Nick was a step closer to modeling? Nick was focused on the behavior of the arrow, talking about how the arrow would travel and about possible changes in its direction. His contribution to the discussion seemed to have been triggered by his reading of the code. From talking about the simulation in general, or talking about visual details of the simulation in particular (that he was doing before), he started to talk about the motion of the arrow represented in code.

In addition, in order to support his ideas, Nick used observations he had from similar phenomena, such as the motion of a gun bullet, that were possibly easier to understand and could be used to support their arguments. A possibility is that Nick assumed that making an analogy to the motion of a gun-bullet would have been easier to show why the arrow should move straight.

Similar to what SC students were seen doing in this study, Nick and Joe were telling the story of the arrow, from the particular point of view of the arrow, instead of simply describing the simulation or the phenomenon. As I indicated earlier, this kind of conversation could more easily lead to a conversation about the mechanism, in the context of creating a single program that would result in all the different steps of the arrow.

Following their conversation about writing a different program, students started working on changing the code to resemble Nick's idea about the arrow. Their short conversation became more technical. Reflecting an effort to modify the code to fit Nick's idea, students talked about lines of code that represented parts of the phenomenon.

As figure 6.1 shows, their revised program had the arrow follow a curved motion, falling slowly towards the ground. It is possible to suggest that this was just a revised program that created a better simulation, a better cartoon about the phenomenon. However, prior to reading the code, Nick did not have any issues with the simulation, and only after he read the code, he indicated that he disagreed with the simulation. Further, the code fixed issues that, even though they did not seem to change the simulation itself, they were incorrectly representing the phenomenon in code. Nick and Joe's revised program was in sync with the direction in which the arrow was moving (seth 90 equals a horizontal direction) whereas in the old program the arrow seemed to be shot horizontally but started moving upwards for a while. This is evidence for a possible shift in students' (or at least Nick's) view of the code not simply as the tool for creating the simulation, but also as a representation of the phenomenon.

Why this is a step towards modeling? The program language had now the potential of being used as a modeling medium. Few minutes earlier students were debating about the arrow's trajectory, trying to "develop" and agree on the story of how the arrow would move in the air. Having agreed on the "arrow's story", students could start a conversation about how to program that story, in a conversation that was about changing the code to "fix" the science represented. Most of the changes in the program were done by Nick, but Joe in line 70 indicated to Nick: "Instead of left [in the first part of the program], just do fd. Don't do left, do fd. No, no, no keep right, keep right, then do fd.", suggesting changes in the code to reflect Nick's idea.

```
to shoot
talkto "a
seth 90
repeat 30 [fd 5 lt 0.5 wait 0.1]
repeat 40 [fd 5 rt 1 wait 0.1]
end
```

Old program

```
to shoot1
talkto "a
seth 90
repeat 70 [fd Powers rt 0.1 wait 0.1]
end
```

Revised program

Figure 6.1. Nick and Joe's revisions

<u>Part 3: Samir started thinking about two dimensions in projectile motion</u>

When Samir joint the group, 20-25 minutes later, I asked Joe to tell him about the new program that he wrote with Nick. Joe talked about the differences in the simulations as results of the two different programs, without any reference to the code in the simulations and how that was representing the phenomena, a possible indication that Joe might have been all along thinking within the visualization frame.

That was not surprising, even if there was a possibility that at some point during his work with Nick, Joe was partly working within a modeling frame. Here, Joe was in a situation where I asked him to present a program to a fellow student who had no idea about the program. In such cases, students tended to talk about the simulation, giving a whole picture of what was going to happen. This was different from their conversation with Nick earlier: telling the story of the arrow in detail was much different from giving an overview of the simulation. The first seeks to reconstruct a succession of instances representing the motion of the arrow in detail whereas the second provides a general picture of the phenomenon (the arrow's trajectory looks like half an arch).

I wanted to turn their attention once more to the code, giving Samir the opportunity to hear what Joe and Nick's prior conversation concerned. Thus, I asked which of the two programs was more appropriate for this situation. The first to respond to my question was Aaron who had just seen the simulations of the two programs. He indicated that the second program (arrow traveling straight) was problematic because the archer would "never get the bull's eye" (line 183), indicating that as a program where an archer shoots an arrow to hit a target, would not work properly. Joe was quick to dismiss his concern, indicating that this was a minor one! By simply changing the target's position, the problem would be eliminated. The real issue was which of the two programs would be more realistic, and Joe indicated that if the archer was aiming the arrow up, their original program was appropriate, but if she was aiming at the target then the new program made more sense.

Samir, then (and later supported by Aaron) indicated a new factor: the results depended on the speed of the arrow and how far it would go before hitting the target. His point was interesting: of course the arrow would fall a little towards the ground, like correctly both programs indicated, but how much it would fall depended on how much actual time the arrow traveled in the air. Samir said that if the target was far (e.g., 200 feet) the archer would not get the target using the second program, because the arrow would drop a lot during its motion. On the other hand, Aaron suggested that if the archer is "a strong enough guy" he could shoot the arrow so fast that it can travel for a while before dropping in the air.

200. **Samir:** it depends, it, depending how far it is. If you are, normally it says it's going, 'cause then it kind of go down a bit, if it hit them the great <inaudible> here and over here, but if you kind of shoot up <inaudible> it be see, somewhat like, um, it will be somewhat like this, because you can't, you be kind of aiming up. You really go, ok, it'll be like this, ou, instead of rating <inaudible>
201. **Joe:** if you aim straight……
202. **Aaron**: depend on how fast
203. **Samir:** it's gonna go down <inaudible> down. But you never shoot like guys like 200 feet away, so it'll be like this, instead of putting here, it hit me all the way down to his leg. Take that as where you're aiming for, like see that's like his stomach.

Samir turned the conversation into a new direction, as he started talking about dependencies. He talked about the dependency of how much the arrow would fall during its trajectory based on the distance traveled and the time of that motion.

Students were now referring to their everyday experiences, in order to talk in detail about the motion of the arrow. The comparison of the two programs helped Samir to identify an important difference: the amount that the two programs had the arrow fall towards the ground while moving. The first program (oval-shaped trajectory) was probably a better option for long distances: in longer distances the arrow would fall more towards the ground and the second program was more appropriate for shorter distances (the arrow would fall with less speed towards the ground while moving).

This was also a conversation about the possible mechanism under study. Students looking at the two different programs started to argue that, how much the arrow would drop towards the ground depended on how far *and* in what speed it would travel, indicating a possible relation of the arrow's vertical motion during flight with the duration of the motion. With obviously no prior experience with studying projectile motions, students attempted to analyze the motion they programmed as such! Samir was attempting to "break down" the motion of the arrow into two distinct motions that seemed to happen at the same time: a horizontal motion (how far it would go) and a vertical motion (how much it would drop). Also he seemed to start thinking about how the two motions do not depend on each other.

The time for dismissal was close and we had to wrap up the conversation. It is important to indicate that the claim here is not that students actually started to have conversations about the physics of projectile motion. However, in this conversation Samir seemed to start thinking in a modeling frame, possibly towards a causal mechanism of the phenomenon. In one way, students were ready to move into another cycle of refinements for their models (programs), specifically using a more analytic perspective about the motion of the arrow. The conversation could have led to the construction of a program that has separate subroutines of those two programs, reflecting students refined ideas. In addition, it is possible that if students were to write such a program, they would have found that they did not need two separate programs for two different distances, but a variable for the distance could have been used to account for both situations.

The last conversation was not a prompted conversation. Students, and specifically Samir, seemed to put himself (and possibly started to help others towards this end too) into a kind of conversation about modeling physical phenomena, possibly as a result of the new programs they were looking at. In one way, it was simply the comparison of two different programs meant to account for the same phenomenon that let to a conversation that could have resulted in refined ideas about the phenomenon. For the first time, students were starting to talk about what was causing different parts of the phenomenon, a direction that could lead into both a conversation about the mechanism of the phenomenon and a model (program) that is a representation of the mechanism that causes the phenomenon.

6.2.2. SC: From a descriptive to a causal model: from multiple rules to a general rule

Modeling in science is the process of developing representations of (physical) phenomena (models). This process is carried out by an iterative process of developing, testing and revising models. As such, modeling is a process of refining models of physical phenomena.

It has been highlighted (Louca & Constantinou, 2002) that one of the advantages of using modeling to learn in science, is teaching the process of model development in addition to refining students' understanding about physical phenomena. For this to happen, modeling media that students use should provide them with tools that can support iterations of developing, testing and refining models. In this section, I describe and discuss an episode from a SC group, where students developed a model for a falling ball. I highlight how students can move from one model to another, by using SC's available tools. Prior to any student work, the same phenomenon was introduced to all groups and students had a class conversation about possible ideas to represent the phenomenon with SC.

Part 1: Creating a descriptive model

When I presented the task, all students agreed that a falling ball speeds up while falling (line 178 below, 186, 193). Zen decided to approach this situation by “marking each vertical point of the ball’s motion with its own speed.” In a sense, Zen (and his group mate Sean) developed a program that included a separate rule for each vertical position of the ball’s motion, assigning each position with a particular speed for the ball. This resulted in a number of rules, each associated with a particular position of the ball and with a particular speed. The following excerpt is taken from the class conversation prior to any group work, while students were talking about their program ideas.

178. **Bryan:** Well maybe, I don’t know whether this is possible, but first once someone toss it, once someone tosses it off, it should go slow and fast as it gain speed, once it comes down. Because it wouldn’t be very realistic, if it, if it goes the same speed over and over once it starts.
179. **Loucas:** so how do we do that?
180. **Bryan:** that’s what I don’t know, how can you make it to go slower and then faster.
181. **Jeremy:** that is the question
182. **Loucas:** so, what…
183. **Bryan:** Oh, I know, I know.
184. **Loucas:** ok
185. **Bryan:** first you could um, ok, get get the um rule bar and then you click on the bar and first you move it one down square, but then the next time you bring it down two squares because then it would go faster, <inaudible>
186. **Jeremy:** so we can take the dude, right, and we can make him kick the ball up in the air, so, so , so it wouldn’t make the ball to go this way and then fall straight down. So if you kicked it up it will look like it, it doing it and then fall, and then we can be, and then we can do Bryan’s idea and make it do one square and two squares and three squares.
187. **Loucas:** so Bryan how would the ball know when to use the rule of one square and then use the rule of the second square, um of the two squares?
188. **Jeremy:** it’s special…
189. **Bryan:** well <inaudible> Instead of making it like, it will, oh here’s a plan I just figured It out. Instead of like once it gains speed it won’t like stay at that speed for a very long time, it will go down faster, so each time you stretch the box farther and farther and so it will go faster and faster and faster.
    […]
193. **Zen:** oh, oh! He like saying, when the, when the vertical 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.

[…]

196. **Loucas:** So we have 2 different ideas. One it's Bryan's that says the first will make the ball to go 1 square, the second, the second time the ball goes 2 squares, … 3 squares.

197. **Zen:** it's the same as him except I'm, I'm 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…

198. **Bryan:** I got it, ok…. and then you go…

199. **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.

(excerpt taken from class conversation, 21 October 2002)

Is this modeling? In one way, Zen decided to make a movie of the ball's motion, that included a snapshot for each position that the ball was "changing" speed. His intention was to recreate the ball's motion on the computer screen. He knew that the ball changes speed while moving, even though he did not show in their program how does the speed change. Zen and Sean's first program included several different rules, each one assigning the ball with a new speed for different heights.

Sean and Zen's program was a sequence of events which happened one after another, by specifying for which vertical position each rule would get "fired" (run) by SC. In each rule, students were specifying how many squares the ball would move. They were not providing any information about what was the ball's speed and what was the mechanism that changed its speed. The first was avoided because rules in SC can easily represent motion. Students were inclined to specify how many grid squares would the ball move, rather than have a variable that would specify that number. As for the rate of the change of the speed, students manually added 1 square in every subsequent rule, and thus in every "tick" of the program, the speed of the ball was increasing by 1.

To create this program, students seemed to have a particular plan in their minds: they were adding 1 grid square per software tick to the speed of the ball. Students also created a program that represented the phenomenon in small pieces, each piece (rule) showing the position and the speed of the ball in every tick of the software. Despite the fact that this might be a simple secession of pictures (that describe the story of a falling ball), students were representing changes in the motion of the ball that happened in sequence.

What was missing from their model was a representation of the relation between the different snapshots, which is the rate of the change of the speed. 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 representation 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

One of the problems that Sean and Zen encountered with their initial model was that it was specifically written to work in one particular situation, and as importantly only if the ball fell from a height of 30 vertical. It did not work when the ball would let fall from 28 or 31 for instance, because there was no rule for motion at that vertical. Additionally, if the ball started falling form vertical 27, it would start with a speed of 3 grid squares per tick.

264. **Loucas:** This rule will fire only if the vertical is 30. Where is the rule for when the vertical is 29? This one? This is for 27.
265. **Zen:** oh, I just deleted the rule that was for 29.
266. **Loucas:** ok, do you want to make one?
267. **Zen:** yea. So,
268. <silence while clicking/typing/looking on the screen>
269. **Sean:** not is working <inaudible>
270. **Zen:** 29, I just need to make one more rule.
271. <silence while clicking/typing/looking on the screen>

272. **Zen:** this is the 29. so click on this, click on this , click on this, click here, and then I go here and 1 down. Thank you very much. Now this will work. This will go second from last. So it goes here…
273. <silence while clicking/typing/looking on the screen>
274. **Zen:** yes. Thank you. Now watch.
275. <silence while clicking/typing/looking on the screen>
276. **Zen:** o, oh!
277. **Sean:** it's stopping at 28.
278. **Zen:** ok. Please! What's wrong with it?
279. **Loucas**: let's think about it a little. Fro which vertical do you have rule?
280. **Zen**: here is for 30, this is for 29, 27, 24, and 20.
281. **Sean:** no it goes 1 when it's 30.
282. **Zen:** yea, exactly, it falls 2 when it reaches 29.
283. **Loucas**: ok. What would happen if you let the ball fall from 31?
284. **Sean:** it will not work!

This led to a conversation that I started 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 to start thinking about the idea of having one rule that would do just this.

297. **Loucas**: so how can we make the program that no matter where you let the ball go, it works?
298. **Sean**: have rules for every vertical!
299. **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?
300. **Zen**: 1
301. **Loucas**: and if I let it go from 31…
302. **Zen**: then it's probably going to have a different speed…..
303. **Loucas**: so what is changing in your program?
304. **Zen**: what do you mean?
305. **Loucas**: what is different between this rule and this rule?
306. **Sean**: the vertical
307. **Zen**: and how much moves
308. **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.
309. **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.

Zen now started to talk about the mechanism of the phenomenon, in the context of a possible 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 showing the phenomenon, 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 the different rules that Zen and Sean had created for the simulation, to start thinking about the mechanism that causes the phenomenon. Looking at what they had each subsequent rule to do, Zen indicated that they could have one rule that would increase the number of grid squares that the ball would move.

Of course, Zen and Sean asked for my help for creating this rule (this conversation was taken in October, rather early in the study), particularly for creating a variable that could account for the number of squares. Then they created a new rule in which SC would add 1 to the variable and then subtract that variable from the vertical position of the ball.

This rule had two advantages over their previous program. First, one "general" rule could account for and create the whole motion of the falling ball. Second, the refined model could also be used with balls that were let to fall from any height.

From a modeling point of view, their new rule included a representation of acceleration (the mechanism that was causing the change of speed), even though, it wasn't clear that students thought about it. However, from this point on, students could be in a position to possibly have conversations about that mechanism, for example, to. compare it with a mechanism that multiplies speed.

Why thinking about the code is important for modeling in science and how that is different from thinking about the simulation? Modeling physical phenomena is the act of developing representations of physical phenomena. The purpose is solely the study of the phenomenon and more importantly the development of an understanding of the mechanism that creates the phenomenon. Having rules that simply assign different speeds to a falling ball based on its position from the starting point is one “model” very different from one that includes a variable that causes a change in the ball’s speed while dropping. The second model provides a possible explanation of the mechanism that causes the phenomenon, giving students the ability to represent in a concrete way ideas such as gravity (which causes the change of the speed of the falling ball), velocity and acceleration. The first program is a descriptive model that creates instances with different speeds for the ball, without providing any possible explanation about what might be causing it.

There is, however, a more important distinction between the two models. The first model seeks to simply create a simulation of the phenomenon that is being represented, whereas the second model can be also seen as the mechanism that causes the phenomenon. Thus, the second model is not just a representation of the phenomenon, is a representation of the mechanism that causes it.

Modeling is about understanding the mechanism(s) that cause(s) the phenomenon and then it is about representing that/those mechanism(s) using the available modeling media. Therefore, modeling requires students to see the act of creating programs as the act of developing representations that include how the phenomenon is caused. I suggest that seeing programming as creating simulations with good depiction, and seeing programming as modeling require different foci from learners. When seeing programming as the process of developing simulations, their focus is on the depiction. On the other hand, when seeing programming as modeling, their focus is on the code that creates the simulation.

6.3. SC: Possible difficulties for modeling dynamics

Models of physical phenomena can be descriptive, simply showing or describing what the phenomenon looks like. Models can also be causal, which are representations of what is causing the phenomenon. Causal models can also differ in the type of mechanism represented. For instance, motion can be represented by a kinematics model (a model that would include a mechanism of the speed change during motion), but it can be also represented by a dynamics model, which would include a mechanism that would explain and cause the change of the speed (it will include representation of the forces that act on the object and cause change in its speed). The models that students created in the above episode were models of mechanics and MW seemed to support representation of such models. However, a model of dynamics could have also be an appropriate model for that case, even though it might have been more difficult for students to think and talk about such a model, especially given the fact that forces are not daily observed.

I use this episode (Annie & Bryan group, 25th November 2002) to suggest the possibility that dynamics may be not be supported by the tools available from at least SC in the study, even though MW also does not seem to have any tools to support representation of dynamics. This is an episode from a SC group, while students were having conversations and were working to represent motions of balloons with helium. Annie and Bryan had already developed a model in SC that had three different balloons traveling with different speeds. Annie and Bryan had previously indicated that the different speeds were caused by the different amounts of helium in each of them, even though that was not represented in their program: their programs consisted of rules assigning different colored balloons with different speeds. In the meeting that this episode took place, I wanted to help Annie and Bryan to refine their program to include a causal relationship between the amount of helium and the balloons' different speeds.

Students' program was based on the idea that the larger amounts of helium in balloons caused higher speed. However, students did not represent that in their program. Identifying that disadvantage of their program sparked a conversation about their programs.

310. **Loucas:** so let's run it. It's really good actually. So remember, so what, what, what, why each balloon goes differently?
311. **Bryan:** cause …
312. **Annie:** of the amount of helium.
313. **Loucas:** ok.
314. **Bryan:** and how many points there <inaudible>
315. **Loucas:** ok, but what causes them to go….
316. **Bryan:** helium.
317. **Loucas:** helium, ok? Now, here is what, when I was looking at your game last night, so if I go here, I can see that the red has helium 3 now. And, but if I, if I made the helium 0 the balloon would still be able to move up there. And now the helium is 0 though….
318. <silence while clicking/typing/looking on the screen>
319. **Loucas:** what do you think about this?
320. **Bryan:** <inaudible>
321. **Annie:** we need to make a rule so that the helium, that the more helium it has, the faster it goes, and if it hasn't any it stops and falls.

Annie suggested that they would need to make a rule relating the amount of helium in a balloon with the balloon's speed. The rule that Annie suggested creating was starting to craft a relation between a variable and an object's behavior, possibly leading to a refined model of the behavior of the helium in the balloons. This would probably be the beginning for developing a program that would include the mechanism of causing the phenomenon.

As the conversation continued, Annie's contribution and Bryan's response sparked a conversation about helium and balloon experiences from their everyday life. Bryan suggested that in real life helium does not necessarily make a balloon go faster, it just makes it get bigger. This was an interesting point: Bryan seemed to be unsure about the speed of the helium balloons, and definitely he was not sure about the relation between the balloon's speed and the amount of helium. He seemed, however, sure about the effect of the amount of helium on the balloon's size.

322. **Bryan:** well, in real life helium doesn't really, yea it makes it go up but it can't really make it go up faster, it would just make the balloon bigger.
323. **Annie:** not necessarily
324. **Bryan:** kind of necessarily…
325. **Annie:** I mean …

Bryan was not comfortable with Annie's idea. Even though he said that "yea, we can do that…" (line 16), he suggested that in real life helium does not necessarily make a balloon go faster, it just make it go bigger.

29. **Annie:** see, but if it had any less helium, and it was running out the it would go….<showing the balloon getting to the ground>
30. **Bryan:** oh, yes, that's true
31. **Annie:** it will like go up to here and then it will go that far <inaudible> and more. 'Cause the more helium it has the higher up it will go.
32. **Loucas:** so what kind of rule are you going to write about that? Can you think about that and I will come back, I need to go around the other groups?
33. **Bryan:** maybe we should like, um like have a start up with an amount of helium 'cause like… well, when the balloons are up you won't have the same amount of helium, for <inaudible> they will all lose helium so, you'll put the helium in like up a 100 but it will subtract really, really fast. Like, …
34. **Annie:** that doesn't make sense 'cause it wouldn't to go like this it would go like <showing the balloon going up and coming down fast>.
35. **Bryan:** when it reaches the top, when it reaches the top… then and then it goes yea….
36. **Annie:** I know but it won't just go immediately down, remember, it goes up, it stays at the top, and then a bird comes by and it <inaudible> and then it starts coming down.

Bryan was indicating that he did not share Annie's experience about the relation between the amount of helium in a balloon and the speed of that balloon. If one thinks about it, there are not many ways to compare balloons' speed through everyday experiences and definitely this is not an everyday "experiment." Children may have experiences with helium balloons, but most of the balloons are more or less of the same size with similar amounts of helium. Also, balloons fly up on the sky usually by accident; they are not usually meant to be let go. In that sense, Bryan had probably no experience of comparing balloons' speed based on the amount of the helium, even though it might have been reasonable for one to think that if the reason that was causing the balloon's motion is the helium, the more helium the balloon has, the faster it would go. That however, was not a direct experience, nor did Bryan seem to have any experiences that could have helped him to make such conclusion.

Modeling is (and should be) a conscious effort of building representations causal relationships between physical values, that not only can provide possible explanations about physical phenomena, but also that it can be supported by observations from everyday experiences. In this sense, even though Annie and Bryan turned to their experiences from everyday life to look for instances that can help them, they were also careful to distinguish between experiences that they had (more helium makes the balloon bigger, line 18) and logical arguments (if the balloons are moving due to the helium, balloons without any helium would fall on the ground (line 29) and the more the helium the faster they would go, (line 31)). In this sense, having to modify their programs to include some type of causal relationship, students started to look for examples that could have supported Annie's proposed mechanism.

As the conversation continued, instead of talking about what helium might cause, Bryan started talking about helium's behavior. He suggested modifying their program so that it would have helium to be subtracted from the balloons really, really fast (line 33).

Bryan was not sure about the relation between the helium and the balloon's speed, but he was certain that balloons lose some helium. That was probably a direct observation he had, and he started talking about that helium's behavior. Bryan was still talking about a mechanism that was related to helium, even though it was not related to the speed of the balloons.

Right away, Annie disagreed with Bryan's ideas, because "the balloon would go up and then come down quickly". She indicated that in their game scenario each balloon "goes up, it stays at the top, and then a bird comes by and it pops it and then it starts coming down" (line 34). It is not clear whether Annie was concerned about the physical mechanism of the helium (what Bryan was suggesting would not resemble what happens in real life) or whether she was concerned about their game scenario (Bryan's suggestion would change their game scenario).

What would a conversation about the relationship between helium and balloon speed look like? The conversation could have taken two different directions. In the first direction, students could have simply considered the effects of helium without necessarily understanding or talking about the exact mechanism that was causing those effects. This would have been a theoretical conversation, with a need for logical thinking that would suggest that if a particular amount of helium in a balloon results in the movement of a balloon in the air, a bigger amount of helium in the balloon would have possibly resulted in a faster motion. This could have been easily turned into a rule for assigning behavior to helium. Each "point" of helium would result, for example, in 1 grid square of position change per tick of the program, as the way of thinking I tried to encourage:

81. **Loucas:** do you want to do that [add a relationship between the amount of helium and the balloons' speed}?
82. **Annie:** yea…
83. **Loucas:** but?
84. **Bryan:** we are confused.
85. **Loucas:** you were confused. Ok, so let's first start with the red one. And if I double click on the red one there is the rule and there is a helium. So, right now, here, the red has 0. If the helium is 0 what should be happening?
86. **Annie:** it would just stay on the ground.
87. **Loucas:** ok, ok. So,
88. **Bryan:** if it's going up <inaudible> it's the air pressure.
89. **Loucas:** so, let's let's talk about helium here. Now, this, let's suppose this has 30 helium you said, and the red has 30? Ok, and because, when it has 30 it's starting going up, ok? And you also said, so why, why doesn't go up. Why does it go up when you have 30 helium?

In a second direction, students could have had a conversation about the actual mechanism of the helium that is causing such motion. In addition to a relationship between helium and speed, a subsequent model could include a representation of the mechanism that causes this particular relationship. This second conversation might have been less theoretical but it would require experience that would support some understanding about the mechanism of the phenomenon. Further, it would also require a scaffolding structure from SC to support rules that represent ideas from dynamics, e.g., forces. During the conversation, students seemed to wanting to understand what the helium was doing to the balloon, rather than simply show the effects of different amounts of helium. In this sense, rather than talking about a "motion model" they wanted to talk about a "dynamics model".

98. **Loucas:** ok. When the helium is 0 what's, what's gonna happen.
99. **Annie:** it will fall on the ground.
100. **Bryan:** <inaudible> won't go anywhere.
101. **Loucas:** and why doesn't go down to the ground, when, when there is helium.
102. **Bryan:** because…
103. **Annie:** helium lives.
104. **Bryan:** and then it loses cause like …

105. **Loucas:** what do you mean? Lives?
106. **Annie:** I am not exactly sure what helium is….
107. **Bryan:** it makes light objects.
108. **Annie:** if it's inside of them.
109. **Loucas:** ok, how? How does helium…
110. **Annie** <gesture that she has no clue!>
111. **Bryan:** like it is some kind of gas, that is, like it's like it wishes like the air pressure and it makes it like rise. Like heat it rises from <inaudible>

Despite my efforts to focus the discussion on what the helium was causing the balloons to do, Annie and Bryan were thinking about how does helium cause the balloon to move. Very soon Annie indicated that she had no clue (line 110)! They might have been sure and specific about operationalizing a definition of what helium does: it makes an object light (Bryan, line 107), when it is inside of them (Annie, line 108), but they were not sure what the helium actually makes that to happen (Annie: I am not exactly sure what helium is…, line 106). Bryan also started to make an analogy with "heat rising" (line 111), even though he was using several terms such as air pressure and gas in ways that was not clear if he was making meaningful uses of them.

During the conversation, Annie started becoming uncomfortable. It seemed as if she had lost interest for the topic and was not sure how to continue. She seemed to be puzzled about what helium was doing to produce the motion in the balloon (line 106), and that might have made her uncomfortable continuing the discussion. In other words, students were starting to think about making rules that showed how helium caused balloons to be "light" and thus fly (lines 12, 34, 60 (indicating to each other that they needed to create the rules for helium), but at the same time they were not sure about what a possible mechanism might be.

At that point I decided to change the conversation a little. Instead of talking about how does helium makes objects light (for which students were not sure), we started talking about what is different between 2 balloons with different amount of helium. In that sense, I was not asking about a mechanism of how helium works in the balloon, but what motion different amounts of helium are causing.

112. **Loucas:** so, ok, so we don't know that, so if, so if I had a blue balloon, a red and a blue balloon, the difference between them is that the blue one will have more helium.
113. **Bryan:** and it will go faster.
114. **Loucas:** so, how can you make a program to show that depending on the helium, because this one has double one, double helium, it's going faster. Because, …tell me.
115. **Bryan:** um, let's say like …each, each one of these [grid] squares are, it's <inaudible> 3, so if you press play and then you press stop this would, the rule would move the balloon 3.

Comparing two different balloons with different amounts of helium did not seem to help students talk about what was different between the two balloons. As Bryan suggested in line 115, it was a simple matter of assigning the balloon with the larger amount of helium, a higher speed, without any need to include a relation with the helium in the rule. I was hoping that the comparison might help them to start thinking about the relation between helium amount and the speed, but Bryan responded thinking only about one of the two balloons. A possibility is that rules are assigning behaviors to individual objects, and thus creating rules about different balloons were different processes.

In their conversation, Annie and Bryan had been use the programming language to refer to balloon's actions (e.g., line 115) and possible relation of motion with the helium (e.g., lines 33). If they were in a "mode" of communicating ideas using SC as the tool and medium of communication, then there was a constraint of talking about what helium does that results in the motion. Talking about how helium *works*, requires a conversation about forces. There is something pushing the balloon towards the sky, because usually, a balloon without helium stays on the ground. It is possible that a conversation about forces could not have happened in the context of SC, because SC programming language does not support representation of dynamics as opposed to representation of motion. That is another way of explaining my intervention in line 112. I wanted students to talk about the motion of the system instead of dynamics, simply because SC is mostly developed to represent motion and not forces that cause that motion.

To talk about how helium makes the balloons to fly in the air, requires a conversation about dynamics (forces). There is something pushing the balloon towards the sky, because usually, a balloon without helium (or with air) stays on the ground. This conversation is a difficult one because students have to use concepts such as forces that are abstract and cannot really be observed in everyday life (students may understand the concept of force, for example, when pushing a toy-car, but not necessarily in the case of pushing a balloon to the sky). Further, students were having this conversation in the context of writing programs (rules) with SC. It is possible to suggest that SC does not have any tools that can easily support representation of dynamics as opposed to representation of motion. Variables such as speed are easier to “invent” because they are representations of physical values and they can be identified during the act of writing programs. Forces, on the other hand, require multiple pieces of information about their magnitude, direction and on which object they act.

Of course, students could simply have thought about identifying a parameter of “helium” and scaled the vertical speed by that parameter, developing a relation between helium and vertical speed. In fact, this is an easy thing to do in SC, because rules can include speed in ways that are “operationally defined”: a rule for a particular object can consist of the direction of the motion, and the amount of grid squares that the object would move (magnitude). However, despite my efforts, and particularly showing children how they could create rules in that way (I showed them how to program a balloon to move 3 squares when the helium was 30), students could not proceed. When I asked them why they wouldn’t proceed they indicated that they were confused as to what the helium was doing.

In the following 5 minutes, the students decided to start working on a different program. They were unsure about how to continue with this one, and more importantly they were not sure what rules to create. A possible interpretation is that the students wanted to develop a model that included a causal mechanism of the phenomenon, but they were not sure about that mechanism. Thus, they decided to work on a different phenomenon!

6.4. MW: Distinguishing between programming and modeling

As I indicated in the beginning of this chapter, the distinction between programming and modeling frame is important. Can students, however, distinguish between using CPEs as programming tools and as modeling media? And more importantly, can they use programming for modeling purposes or just for creating simulations that show physical phenomena?

In the short excerpt of student conversation presented in episode MW4 (see the transcript of this episode in chapter 3), Richard and Aaron had just finished debugging their two programs: one for a person jumping on the Earth and another for an astronaut jumping on the Moon. Their programs consisted of a number of instructions that represented the motion of each person jumping. Their programs consisted of two parts, one representing the upwards motion when jumping and the other was representing the downwards motion of the person jumping (see figure 6.2 for details of their code). However, students did not explicitly include aspects such as the mechanism for the change of each persons' speed while moving in the air. Rather, their programs were sequential representations of what students thought would happen in each "moment" of the person's movement: first she would move fd10, then fd9, then fd8 etc. In other words, students had developed descriptive models that included a sequence of instructions about motion of the characters, with one instruction following the other with changes in the speed.

Students then considered the possibility of modifying their programs to make the differences of the two persons jumping easier to be observed in the simulation. That sparked a conversation about how their programs looked and what possible changes to include. Even though the conversation started from looking at their simulations, very soon students started talking about the code of their programs.

Richard disagreed with Aaron's program (Aaron had typed the first two programs), because Aaron's program was showing the boy on the Earth jumping much slower than the astronaut on the Moon. Richard was suggesting that the astronaut should jump slower than the boy on the Earth, showing with gestures how that motion should be. Aaron, on the other hand, was thinking that because of the greater gravity on the Earth, their program should show the boy jumping slower.

```
to jumpmoon
talkto "t1
fd 8 wait .2
fd 7 wait .5
fd 6 wait 1
fd 5 wait 2
fd 4 wait 3
fd 3 wait 4
fd 0 wait 3
bk 3 wait 2
bk 4 wait 3
bk 5 wait 4
bk 6 wait 5
bk 7 wait 6
bk 8 wait 7
end
```

```
to jumpearth
talkto "t2
fd 5 wait 4.4
fd 4 wait 3.2
fd 3 wait 2.4
fd 0 wait 2
bk 3 wait 2.4
bk 4 wait 3.2
bk 5 wait 4.4
end
```

Figure 6.2. Aaron and Richard's programs

The conversation quickly turned into talking about specifics of their code, talking about what their current code was showing and suggesting possible alterations. Their focus was specifically on the "wait" primitive that they had in their programs and they were disagreeing about which of the two programs should have the more wait. However, talking about what their program should have been, did not seem to be a useful conversation, at least for thinking within a modeling frame about the phenomenon. Students seemed to agree on the difference of gravity in the two situations, but they were disagreeing on the effects of the difference of gravity for a person jumping on the moon and on the earth. Because of that they did not seem to agree on the changes for their code, too. Aaron and Richard were talking about the specifics of the code, making references to their sense of how the real world works, but they weren't using the code as modeling, because it was simply used to show what happens.

It is important to note that prior to this conversation, and before deciding to make a change in their code, students did not have any specific issues with their programs.

However, they were using the code to simply create a simulation of two characters jumping on different planets. A discussion about phenomenon representation with code, seemed to support a conversation about the use of code as a modeling tool, representing the phenomenon. Soon, Aaron made an interesting move in the conversation:

244. **Aaron:** [to Richard] Stop talking in the computer language! Does the moon have way more gravity? I mean less gravity?

Aaron tried to change the focus of the conversation, from the code to the phenomenon itself. He seemed to want to talk about the actual mechanism that was causing the difference in jumping on the two planets, before continuing the conversation about the code.

What made Aaron change the conversation? Maybe he sensed that the conversation was about changes in the code that reflected the causal mechanism of the phenomenon, that he identified in line 233 as gravity. Students were talking about those changes without talking about the actual mechanism that was causing the phenomenon. Before moving on, Aaron wanted to have a conversation about that mechanism, and more specifically about gravity. The conversation, however, that they had thus far was within a visualization frame, as they wanted to improve the depiction of their simulation. Thus, Aaron made the move to take the conversation away from using the code as a visualization tool, and talking about gravity on the moon and on the Earth, seemed (at least to Aaron) to be an appropriate choice.

Of course, students could have had a conversation about the mechanism underlying the phenomenon using the program language, as I had seen (and discussed in this chapter) in other cases. However, in this case Aaron wanted to have a conversation about what was causing the phenomenon, in order to resolve the differences they had about what changes to make in their program. At that point in the conversation, students were using the program language as a tool for creating a simulation (working within a visualization frame), and the conversation that Aaron wanted to have was about the mechanism that was causing the phenomenon, and needed an approach from within the modeling frame. For this reason, Aaron probably wanted to make sure that the conversation would be about gravity and not about code as a representation of the phenomenon or about the code that creates a simulation.

That Aaron made a move initiating a conversation about what ultimately made the difference between jumping on the moon and on the earth, was a move towards modeling for at least two important reasons. First, he wanted to have a more useful conversation that would help students to make progress in developing some agreement about the representation of the phenomenon. For this reason he chose a conversation about gravity on different planets and its effects on jumping, which was a conversation away from talking about or with the program language.

Second, because students (like Aaron here) can realize the need for such a conversation, then teachers can have possibly prompt them to turn their program into a representation of gravity (which would have created a causal model instead of a descriptive one). Students were already having a conversation about what was causing the phenomenon, and using those ideas to develop a representation of the phenomenon would have probably been an easy step towards modeling.

6.5. MW: Code as the representation of the mechanism of the phenomenon

In this episode, two groups of students working with MW were sharing their programs with each other. I have isolated the conversation that they had about the program that Tilson and LJ started to develop, about a person walking on a moving train, in the direction of the motion of the train. Tilson and LJ wanted to make a scene in which a person would arrive via helicopter on the train, and then walk towards the front of the moving train. Because LJ did not make it to the club the time before the meeting, all of them (but Tilson) had never seen this program before.

In describing and analyzing this conversation, I focus on the use of code as a medium for communicating the mechanism that underlies relative motion. As in the previous section, students here started talking about the mechanism, but unlike Aaron and Richard, Jiana started talking about what was causing the phenomenon by using code.

At the point that the conversation took place, the program was incomplete: the train was following a program that had it accelerate, then decelerate and then stop, but the boy was programmed to simply stand on the train (that is to move with the same speed of the train so that it would look like it was moving with the train). Because their goal was to make the person walk on the train, during the previous meeting Nick and Tilson had also programmed a routine which was animating the boy so that it looked like he was moving his legs while walking.

Among the first things that students in the study (and in this case) were inclined to do, was to run the program and see what was about. This was definitely a much easier way to figure out what the program was about, rather than reading the code and trying to figure it out. Also, it is much more fun to see the animations, and the different characters' behaviors, rather than trying to figure them out through reading the code. However, when students turned to talking about the code that created the simulation, the conversation became much more focused and specific. Students could be specific about things like speed, that it was difficult to figure out from the simulation itself. That conversation also helped students to focus on the mechanism that was creating the phenomenon.

When they first saw the simulation, students were focused on issues like the size of the train, the animation of the person walking, and the position of the boy on the train. They were also focused on the simulation itself, talking about what it seemed to represent.

26. **Loucas:** […] So, what do you think the boy is doing on the train, right now? What it looks like?
27. **Jiana:** it looks like he's riding.
28. **Gabriella:** he's walking in his place.
29. **LJ:** walking in place
30. **Jiana:** I think he should be like going forward.
31. **Loucas:** what do you do?
32. **Gabriella:** I'm trying to get the um…this…
33. **Jiana:** either the train can be longer and he can be walking forward.
34. **Gabriella:** yea, it looks like he's on the, you know one the treadmill or something. That look's like a train, a big treadmill that looks like a train.
35. **LJ:** but he's going slow…
36. **Loucas:** I don't understand that (what G said).
37. **Gabriella:** because he said, he's walking …
38. **Jiana**: it's like a train slipping out under him; he's walking along the train
39. **Gabriella:** he's going like this.
40. **Jiana:** cause as long as he was one foot down, um he'll be staying on the train. Because it's um like if he jump off, the train would go further ahead of him but because he's walking he'll be going further along the train.
41. **Loucas:** ok. But what I'm asking is what *exactly* is the boy doing.
42. **LJ**: which one is the boy [t1 or t2]? <LJ switches to program window>

Students in the previous excerpt were quick to identify that the boy on the train was doing something unusual. He seemed to walk in place (Gabriella, line 28), riding on the train (Jiana, line 27) or the train seemed to be slipping under him (Jiana, line 38) or he was on a treadmill (Gabriella). All interpretations of what was going on in the simulation were accurate, even though they were interpretations from different perspectives: the boy's perspective who seemed to walk in place and the train's perspective which seemed to be sliding under the boy.

When I asked students about ways to "fix" the simulation, students started providing general ideas, such as "the boy needs to move forward" or "the train needs to be longer" so that the boy won't fall while walking.

It was only when I asked once again what exactly was the boy doing on *the* train that LJ went into the program screen and started reading the code that created the simulation. Now students started a different kind of conversation. Instead of trying to guess what the simulation what doing, they were trying to find the part of the program that was about the boy and then figure out what the program had the boy doing. In that sense, students probably started to see the program as the mechanism that was causing the behavior of the boy. They spent some time figuring what each routine of the program did and then they tried to figure out what the boy was doing on the train.

244. **Gabriella**: the train is t2?
245. **Loucas**: yea.
246. **Tilson**: oh, you mean right here?
247. **Loucas**: no that's to set place…
248. **LJ**: oh!
249. **Gabriella**: But it doesn't say, um it doesn't say that here how much the cat equals.
250. **LJ**: same speed…
251. **Tilson**: it adds 2!!!!
252. **LJ**: No it doesn't!
253. **Gabriella** [the speed is]: 2
254. **LJ**: no it doesn't! See, is it…
255. **Tilson**: yea it does.
256. **Loucas**: guys, hold on. Why do….
257. **Gabriella**: it goes by 2
258. **Tilson**: 2 4 6 8…

[…]

279. **Gabriella**: […] Well, whatever it still goes by 2s. it goes 2, 4, 6, 8, 10, 12, 14, 16, 18.
280. **Jiana**: and then it subtracts.

Their effort to figure out what code was doing, had two characteristics that were unique for their conversation. First, that for most of them this was the first time that they saw the program, it probably helped students to closely read the code and try to figure out what it was doing. Students were not concerned about the simulation any more, even though they were using it to understand the code. For them, the code was causing the simulation: for instance the train was moving and students expected to see a program primitive like “forward <something>” for the train to move.

This characteristic was an important distinction between their conversation previously and their present conversation. Students were not trying to guess what the boy was doing any more. They knew that the “mechanism” of what the boy was doing in the simulation was somewhere in the code and they were looking for it. In many ways, this distinction between the two conversations (trying to guess from the simulation and try to read from the program) is a distinction between describing a phenomenon and trying to figure out what was causing the phenomenon or how it happened.

Second, the program that Nick and Tilson started to write included code that created relationships between subroutines. For instance they were using a variable for the train’s speed (which it started from 0, it increased for some time and the decreased to become 0 again), which they were also using for the motion of the boy. Even though the boy was not actually walking on the train, for the boy to stay on the train, they had to write code to move the boy with the train. This added a difficulty to students who had not seen the program before, probably because it was more complicated than a simple program: the boy was moving forward as much as the train was and thus, the boy in the simulation was staying on the train.

As soon as they figured out what each line of code was doing in the program, I asked them once more what we needed to do in order to make the boy move on the train, as LJ's original idea was.

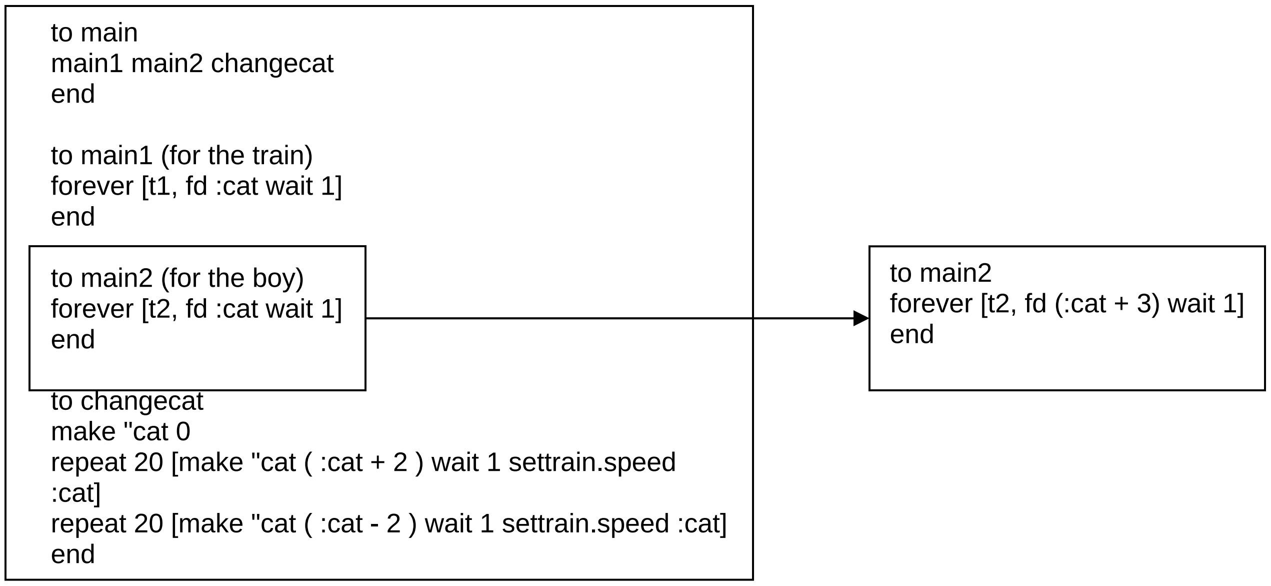
This context put students into a situation where they had to modify code to fit the science of the phenomenon represented. As I previously noted this seemed to be the most productive context of modeling conversations, where students were using the code as a very specific design medium to re-create a phenomenon. Equally important, the program that was required for representing the relative motion in this case, was also the mechanism of the phenomenon and it required a detailed understanding of what was going on. Students for instance had to understand why when a car moves next to a truck, it sometimes feel like the car is moving backwards (as Gabriella indicated in the conversation).

Students were quick to identify that they needed to have a fd amount for the boy to move, but they also considered the fact that the speed of the train was changing all the time and thus a simple number assigned to that fd would not work. This was making the program more complicated. They also noticed that the boy already had a fd that was the same with the train. For that, Gabriella indicated that they should add some more so that the boy would move fd. Jiana then made a nice interpretation of the situation (referring to the code shown in figure 6.3): the train was pulling the boy (that's why the boy was moving fd the same amount as the train) and in order to walk the boy had to pull himself too!

> **Jiana:** because, if um, if he's going same speed [with the train], he's just gonna, it's kind look he is doing this again [staying in place on the moving train]. And if he's doing that, that's not right.

‘Cause he is um actually moving fd like his <inaudible>it’ll go fd and he should be pulling himself along, but instead he’s doing this! […] ‘cause the train is pulling him and he is pulling himself too.

Jiana’s interpretation of the situation was both a programming suggestion and an explanation of relative motion. Interpreting the situation into a programming suggestion, Jiana gave a description of the relative motion from the point of view of the boy on the train. This was an important step, because as I have indicated before, talking about system changes is not productive for developing representations of phenomena with CPEs. However, talking about particular objects in the phenomena (like the boy and its motion) could be much more productive.



```
to main
main1 main2 changecat
end

to main1 (for the train)
forever [t1, fd :cat wait 1]
end

to main2 (for the boy)
forever [t2, fd :cat wait 1]
end

to changecat
make "cat 0
repeat 20 [make "cat ( :cat + 2 ) wait 1 settrain.speed
:cat]
repeat 20 [make "cat ( :cat - 2 ) wait 1 settrain.speed :cat]
end
```

```
to main2
forever [t2, fd (:cat + 3) wait 1]
end
```

Figure 6.3. The code that Jiana was referring to, and the subsequent modifications

More importantly, however, is how Jiana’s contribution was also a contribution to developing understanding for the relative motion. In this case, the code was used as a communication device of that understanding in the context of trying to modify a program in MW. In a way that seemed to make sense as both a programming suggestion and as a possible analysis of what was going on in relative motion, Jiana described a possible change in the program. The code was used to clearly represent the mechanism of the phenomenon.

6.6. Summary

The purpose of this chapter was to provide a description of the three proposed “frames” that seem to describe students’ work (conversations, activities, focus and work goals) with MW and SC, provide descriptions of shifts of students’ work towards modeling in science by using the CPEs and describe episodes of modeling work and conversation with CPEs to indicate several promising possibilities for using CPEs for modeling in science.

In this chapter, I have also provided some description about possible shifts in students focus during their work, based on the micro-context of their work (e.g., specific goals, ways of interpreting questions etc). Based on the data presented, I have argued about the promising possibilities in which CPEs can support students’ thinking and work with modeling in science. MW, which uses textual-based language, can possibly serve as a modeling medium that students can use to read and write code as a way of representing the mechanism of what causes natural phenomena. In sections 6.2.1 and 6.5 I have highlighted two instances in which some of the students saw and interpreted the code of the simulation to cause the simulation and subsequently the phenomenon it represented. In section 6.2.2 I have presented an episode which suggests that SC, which uses rules that assign objects with behaviors, could possibly serve as a tool for making generalizations to objects behavior, by representing a sequence of otherwise related behaviors (such as a behavior that keeps changing) and helping students to see ways to create rules that cause the sequential behaviors.

Shift in student focus and thinking in the context of working with CPEs in science may be related to shifts in student thinking (Louca, Hammer & Bell, 2002) and shifts in students' views of knowledge in science (Louca, Elby, Hammer & Kagey, in press). In this study, students work and focus changed in a matter of minutes, probably based on the context of students' work at those particular moments. Students who were using CPEs as programming or visualization tools, were seeing to change focus from simply viewing code as a tool for creating simulation that shows what happens in the phenomenon, to using code or rules as the representation of the (mechanism of the) phenomenon. In modeling this implies a difference in the type of model that students created or started to create: a descriptive model (before) and a causal model (afterwards). For cognitive science this implies that students *have* abilities for using program code as the representation of the phenomena, despite the large amount of data in this study where students were seen to use CPEs as visualization tools.

Viewing students as having abilities that are (micro)context dependant, is very different from views of children's abilities as developed in concrete stages (developmental views). In other studies (Louca, Hammer & Bell, 2002; Louca, et al, in press) researchers have started crafting a possible usefulness in the first view, particularly in respect to teaching practices. Rather than seeing students as needing to develop abilities, seeing them as having abilities to for example use program code as a representation of a natural phenomenon (as data from this study may suggest), or for argumentation in science (Louca, Hammer & Bell, 2002), or that students are able to respond to explicit epistemological instructions (Louca, Hammer & Kagey, 2003) has important implications for instruction. An early agenda in science education is to help students come to enter and work within more sophisticated modes of scientific inquiry more reliably, to see them as part of what “doing science” entails, and from there to develop greater facility. Rather than expect early development in scientific inquiry means forming new abilities, educators might better see it as a matter of applying and refining abilities students already have.

Finally, findings from this descriptive study, as presented and discussed in this chapter, do not seek (and cannot support) claims about student modeling, partly due to the small duration of the study and to the small number of students involved. Rather, findings suggest promising ways that CPEs can be used to support student thinking for modeling in science.

7. IMPLICATIONS FOR SCIENCE TEACHING AND SOFTWARE DEVELOPMENT

The purpose of this study was to investigate and document the use of two computer-based programming media (CPEs) (Microworlds Logo (MW) and Stagecast Creator (SC)) by fifth grade students for modeling natural phenomena. The central research question focused on how might CPEs support fifth grade student inquiry in science, and specifically (i) how fifth grade students might use MW and SC for developing models of natural phenomena, (ii) what were the characteristics of student thinking that were supported by the two CPEs used in the study and (iii) what were the characteristics of those CPEs that (could) support collaborative modeling among fifth grade students in science.

In chapters 3 through 6 I have presented and discussed findings for the three above subsidiary research questions. In this chapter, I summarize findings and their implications in a way that can be useful to science educators and software developers. I discuss issues related to how modeling with fifth graders can look like and in particular how productive thinking and productive conversations can look like for modeling among young learners. I also summarize possible relations of findings with the characteristics of the software that (could) support collaborative modeling practices among fifth grade students and implications of those software characteristics for building models of and learning about natural phenomena. I start with a brief summary of students' activity and conversation patterns with each of the two CPEs.

7.1. Microworlds Logo: Students' activities and conversations

During planning, students working with MW primarily talked about the structure of their program, breaking natural phenomena down into small programmable pieces that share common characteristics, particularly in the way individual phenomena looked. During that time, students used the program language as a communication medium to talk about how their simulation would look in order to support their programming decisions. During writing and debugging their programs, however, students working with MW focused on writing programs. Their conversations were limited and strictly technical, i.e., about the code and program primitives. In a different "mode" of conversation, after their programs were successfully running, students were reading code and using their programs as representations of natural phenomena. During that time, students used the code of their programs (and not the resulting simulations) to talk about the representation of a phenomenon in code.

Early in their work with MW, students dealt with the characters (objects) and the background(s) of their designs. During writing and debugging code they were focused on getting a program to run successfully without paying any attention to their program's structure which they had discussed during planning. After getting their program to run successfully, students changed their focus to modifying and improving depicting details and fine tuning the details of their simulations. Lastly, when students modified code to match, for example, a slightly different phenomenon, they focused on changing the science that was represented, which allowed them to see the program as a representation of the phenomenon rather than seeing it as the tool to create a simulation. Students read their programs in detail, tried to identify what each line of code represented, in order to make appropriate modifications. In this way, students did not deal with any technical issues, but they were thinking of ways to represent the phenomenon in code.

7.2. Stagecast Creator: Students' activities and conversations

During planning of their work, students using SC talked about the overall story of their games in detail, describing a succession of events that would happen in sequence, one after another. During programming, students tried to translate their story into rules. They were focused on programming and used the program language in their interactions, which were more extensive than the conversations that MW students had during programming. Students working with SC were also making some references to experiences and observations of natural phenomena from their everyday life to support their program ideas.

Early in their work with SC, students dealt with the characters (objects) and the background(s) of their designs, spending much more time on how their designs looked than the time MW students did. Debugging was in the form of deleting rules and making new ones. While creating and debugging their rules, students working with SC were focused on translating their story or game into rules; thus translating their game plot into programmable pieces in SC. They were focused on creating a simulation that would depict reality as well as possible, in addition to creating "cool" games that met several criteria (i.e., visualization, story plot, scoring system, levels of difficulty). Lastly, in a few cases, students improved their programs by replacing their rules with more general ones (and subsequently more complex). Rather than having multiple rules subsequently assigning the object with different states of behavior (i.e., different speeds), SC students replaced them with fewer rules that affected/altered the properties of the objects (i.e., the speed of an object) see discussion below and section 6.2.2 for more details).

7.3 Possible advantages and disadvantages of Microworlds Logo and Stagecast Creator

7.3.1. Microworlds Logo

A large part of MW students' work and conversations was technical, concerning the program primitives and structure. Conversations about the structure of their programs put students into a mode of thinking about the code, instead of thinking of code as a tool for representing natural phenomena. This had both an advantage and a disadvantage. The disadvantage was that students entered a technical mode of work, during which they were concerned only with creating programs that ran successfully.

On the other hand, however, from as early as their planning sessions, students working with MW started to think about possible ways of breaking up their phenomenon into pieces, based on shared characteristics of parts of the phenomenon. Modeling in science is developing representations of the natural phenomena that include the mechanism of what is causing the phenomena, and similarities among object behavior(s) in a phenomenon can help students to focus on what is causing that behavior(s) and how to represent it. Code, like mathematics, can be used as a way to represent natural phenomena clearly and precisely. Additionally, code, unlike mathematics, can be understood more easily as the cause that actually creates (not simply shows) the phenomenon represented in a simulation (Sherin, 1996). Breaking down phenomena based on object behaviors and what is causing them can be a step towards scientific modeling.

During writing and debugging their programs, students working with MW tended to work through a “programming frame,” focused on writing programs. Their actions and conversations were about the code and the correct program primitives, putting them into a “technical mode of work.” This had both advantages and disadvantages. The disadvantage was that this technical mode of work seemed to shift their focus from the program’s structure, to the syntax of their program, resulting in early programs that consisted of a single routine that had little to do with students’ initial plans. Rather than using their plans and focusing on writing programs that would represent a phenomenon, students created programs that could run successfully and create a simulation that looked as realistic as possible.

On the other hand, working through a “programming frame”, students saw their work as writing, testing and correcting code, which is useful for scientific modeling. Thus, students working with MW started using iterative debugging which is another feature of scientific modeling. However, the technical mode of work shifted their attention to a different objective - instead of focusing on the representation of science, they turned their attention on technical details of their code: they used their program to depict reality instead of developing representations of natural phenomena in code. This was a disadvantage.

Modifying code to match, for example, a slightly different phenomenon or idea, seemed to be a more productive context for modeling in science. Students were now working from a different frame, with different characteristics and focus. To modify their code, students had to read it carefully and identify what each line of code represented. In this way, students did not have to deal with any technical issues to make their program run – but, they had to think of ways to represent the phenomenon in code. Therefore, unlike the “programming” and the “visualization frames”, the act of modifying one’s program put students into a productive mode of work for scientific modeling (“modeling frame”).

7.3.2. Stagecast Creator

During planning, students working with SC talked in detail about the overall story of their games, describing a succession of events that would happen in sequence, one after another, with emphasis on creating “a cool game.” Their programs usually consisted of a large number of rules (as opposed to a single routine that early programs in MW consisted of) that were meant to be executed by SC sequentially. Despite the disadvantage of using a “non-linear” CPE (see discussion in the following sections) in a “linear way” – which resulted in thinking about the system changes while having to program object behaviors – there might have been an advantage. A descriptive program of an accelerated motion, for instance, that one group developed with SC (see section 6.2.2 for details) included a number of rules that assigned a falling ball with higher speed in each subsequent rule. This was not modeling the phenomenon because it provided no information about what was causing the changes in the ball’s speed. It provided students, however, with the opportunity to realize the relation between the subsequent rules and replace them with others that would re-create each “scene” – which is a representation of the mechanism that underlies the changes in each subsequent rule. Just as figure 7.1 shows, it is possible to realize that what is changing in each subsequent rule is the distance that the ball moves (which is the relationship among the rules) and create (a) a variable for the distance that continuously changes, and (b) another rule that instructs the ball to move that distance.

During creating and debugging their rules, students working with SC were focused on translating their story or game into rules that could be used to program a simulation. They were concerned about depiction in their simulation, in addition to meeting several criteria of good computer games (e.g., visualization, story plot, scoring system, levels of difficulty).

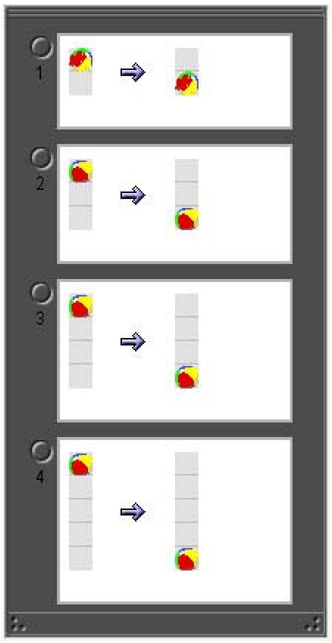


Figure 7.1. Example of rules for a falling ball in SC

Modifications of descriptive programs were also a productive context for modeling in science, because students could talk about a visually represented program and decide about ways to improve it. In one case, improvements started in an effort to make their program more general which led to creating a single rule that affected the speed of the object, rather than having a rule for each different speed value of the character (figure 7.1). In this sense, students started to work toward a direction of representing the causal mechanism of the phenomenon.

7.4. Implications for science education: Modeling in science with young learners: how does it look and why is it important

This study was based on the idea that programming can be used as a tool for scientific modeling, that is, for developing models of natural phenomena (diSessa, Abelson, Ploger, 1991; Louca & Constantinou, 2002; Redish & Wilson, 1993; Sherin, 1996; Sherin, diSessa, & Hammer, 1993; White & Fredriksen, 1998; Wilensky & Resnick, 1999). Learners can use CPEs for developing representations of natural phenomena in the form of programs, which can produce simulations of the natural phenomena. Throughout this study, the focus was neither on the students' knowledge (or ideas) nor on the knowledge that they gained. Rather, the focus was specifically on the ways that students constructed models of natural phenomena with computer-based programming tools designed for young learners and the types of programs that they created. In this sense, the focus was on students' activity and conversation patterns, as well as on the way they viewed the programming process as expressed through their work. This kind of focus can be of particular interest to at least two communities: (1) to teachers, who might be interested in what fifth graders' modeling can look like and (2) to software developers who might be interested in the features of CPEs that are more appropriate (or productive) for young learners, specifically for scientific modeling.

In previous chapters I have documented the different ways that students in this study used different CPEs, talked about their programs/models, the different ways that they saw their work with CPEs, and the different frames within which they worked while developing representations of natural phenomena. Those findings have implications for teaching and learning in science, because this study primarily involved documentation and analysis of actual student work with computers as tools for learning. For this reason, in this section I highlight findings that are related with what modeling can look like in science.

Through this study, it has been partially demonstrated that modeling started when students' focus was on developing programs that include code that causes changes of the behavior of the different "objects" involved in the phenomenon. This was observed during shifts of student focus from creating *descriptive* representations of natural phenomena to creating programs that *caused* the phenomena. In fact, during this study, students did not always see their work as developing representations of the causal mechanism(s) of the natural phenomena (also see section below about the different ways that students used code in the study). Thus, having a stance of seeing students as being able to work within different frames, requires that a teacher should be able to detect collaborative modeling practices from practices that are not supportive for modeling and promote them. Teachers should be looking for instances during which students take this particular stance, that is, seeing code as a tool for developing programs that create a phenomenon rather than creating programs that show different "scenes" of a phenomenon. For this reason, I have gathered below features of modeling with young learners, taken from instances from the study during which students were modeling, instead of developing simulations, writing programs, etc.

7.4.1. Focusing on the objects' behaviors and on behavior changes

Modeling natural phenomena by developing programs that are representations of the phenomena, requires that students break down their knowledge about how the system functions into small pieces that are programmable (Constantinou, 1996; Louca & Constantinou, 2002). Students in this study, at some point in their work, undertook this task. Students working with MW started planning their work by talking about the structure of their program (i.e., the different pieces of their program) that reflected different parts of the phenomenon they were modeling. Students working with SC, however, carried out this task later when they were actually developing rules in SC translating their story or game into rules of behavior for each object in the game.

Students working with MW planned their work by breaking down the phenomena into programmable pieces in which objects had similar behavior. The subsequent development of programs was partly reflecting their planning, even though students were not representing what was causing those changes in the objects' behavior. The latter was common in this study and has important implications for young students learning modeling: students working with MW and SC initially did not see their programs as representations of the mechanism that was causing the change in objects behavior, but they saw them as tools for creating simulated representations of natural phenomena. In this sense, they developed lines of code or created rules that reflected the objects behavior without any representation of what caused the changes in that behavior. In some cases in MW, different lines of code were representing altered behaviors, whereas in SC subsequent rules were causing different behaviors. In both cases there was no representation of what was actually causing those changes.

The important point here is that in this study, given a new situation/phenomenon to program, I anticipated that students would develop programs descriptive of the object behavior and not descriptive of the mechanism(s) causing those changes (see later discussion about descriptive vs. causal models of natural phenomena). One interpretation is that modeling natural phenomena with young learners starts with the development of descriptive models of the phenomena, before students can read, evaluate and subsequently revise their models to become causal (see discussion about descriptive and causal models below).

For the latter to happen, however, it is important for students to be able to read their programs, seeing them as representations of the phenomenon's underlying mechanism. In this case, the simulation simply shows what the phenomenon looks like. Having different lines of code represent different object behaviors can help students to see or identify relations between different subroutines. In turn, a descriptive model of a phenomenon can help students to talk and subsequently develop a causal model of the phenomenon by writing code that actually creates those different behaviors. Students using MW and SC were observed in this study to do so, even though reading code in MW seemed to be more tangible than reading rules in SC (see discussion below about different uses of code). In the study, shifts towards developing causal models happened in the context of reading and talking about the programs themselves and not about the simulation that then resulted.

7.4.2. Telling a story….

Students working with SC saw their work as developing games that reflected or followed some natural phenomena, e.g., balloons with helium, objects falling due to gravity, etc. Telling the story of their game, and subsequently trying to create their game on SC was not productive for modeling in science, partly because by telling a story students were focused on the system behaviors and changes whereas modeling with SC can only be done with programming individual object behaviors (see later discussion about object oriented programming, and write a story vs. write a program).

For SC students, telling a story of individual objects or causal agents (such as velocity, acceleration, hunger etc.), however, and subsequently focusing on what causes the changes of the object behaviors was more productive for developing models of natural phenomena. That is, it was more productive for modeling in science to "tell the story" of acceleration, which was altering the speed of a dropping ball, rather than having a number of subsequent rules that assigned different speeds to the dropping ball (see example in section 6.2.2). For this to happen students did not have to change their particular view of seeing SC as a tool that would represent a story. Rather, they were changing their focus, from telling a story of the overall phenomenon (system) to telling a story about causal agents including how these were causing the objects' behaviors. Teachers can help students use this telling-a-story approach for talking and representing causal agents in their models and thus use a non-productive approach in one case (for system behavior and changes) in a productive way in another case (for talking about causal agents).

7.4.3. The importance of reading the code for modeling

As I have indicated before, the process of model development and model deployment can be compared to the process of writing and implementing a computer program. In this sense, modeling natural phenomena can be carried out through a computer program, when the program itself becomes the scientific model. Developing a model is similar to writing a program using CPEs; model deployment is similar to testing a program and watching the simulation. The program language becomes the design medium for the scientific mode and the program (outcome) becomes a way of clearly articulating one's understanding about scientific phenomena. The simulation only supports modeling (programming), providing an immediate way of testing one's model (program).

Students in this study were often observed to work through a visualization frame, during which they used the CPEs to create simulations that were visually pleasing and looked like observations from the real-life phenomena. Working through a visualization frame, students used the code as a tool to create and refine the visual characteristics of their simulations and their programs did not represent their ideas about the structure of their programs. Their programs were simply descriptive of the phenomenon and not descriptive of the mechanism causing the phenomenon.

Working within a programming frame, students were focused on how to represent the phenomenon in code. They read, talked about and modified their code. Because of this, students started seeing the code as a representation of the phenomenon. In the case of MW, students read lines of code (which were instructions for the different behaviors of the turtle), and talked about how and why to change code to represent the phenomenon better. In this sense, code can be a powerful means for modeling in science, because it is a precise way of representing one's ideas about a phenomenon.

Working within a modeling frame, students in the study were debating about replacing code with more appropriate code, despite the fact that both resulted in the same simulation. Their focus was on what the code was saying and whether that was representing the phenomenon, rather than creating a simulation that looked “ok”. Debates about jumping higher vs. jumping faster as a result of the difference between the gravity on the moon and the gravity of the earth, reflected exactly this: what was the precise mechanism causing the difference in the behaviors (see episode MW4 and section 6.4).

When students used the code itself to represent a phenomenon (instead of simply using code as a tool to create a simulation of the phenomenon), their programs started to become causal models of the phenomena. Students replaced lines of code that represented different “scenes” of the phenomena with code that actually created those “scenes” by causing changes in the objects’ behaviors. Thus, the code was becoming the actual mechanism of the phenomenon. For instance, as presented in section 6.5 reading the code made Jiana indicate that a boy on a train was moving along the moving train because in the code, the train’s speed was affecting the boy’s speed. This made her indicate that for the boy to move on a moving train, he had to pull himself in addition to the train pulling him. This was a result of carefully reading the code about the motion of two objects and interpreting how the speed of the train affected the motion of the boy on the train.

Reading and making subsequent modifications in the code can lead to iterations for model refinement, a feature of scientific modeling. Working within the modeling mode, students in this study justified changes they were proposing for their programs by talking about similar phenomena where the same behaviors or what was causing them were easier to see. This was useful for modeling, because students were justifying the need for changes in the code. For example in episode MW 7 Nick indicated that the program he was reading was not actually representing the motion of an arrow in the air but rather the motion of a rock that was thrown. Because this difference was not reflected in the simulation, Nick was suggesting that what the code was reading was a different kind of motion than the one the program was meant to represent. And he went on to explain why the program was representing the motion of a rock.

7.4.4. Different uses of code: code as the mechanism that causes the phenomenon vs. code that creates a simulation

Students in the study used the code (the textual programming language (MW) or the rules (SC)) in a variety of ways. When they focused on the code itself, students were using the code to write programs that would run (working through a programming frame - see episode MW5 for an example, during which Aaron was typing their first program and Richard was making suggestions about the primitives he used). When they focused on creating a simulation, being particularly interested in depicting details in their simulation, students were using the code to create visual representations that realistically depicted natural phenomena (see episode SC2, where Annie and Bryan had a conversation about how their simulation looked, what details to change to make it look better as a game, and represent the game's underlying story). Lastly, working through a modeling frame, students used the code as a way of representing natural phenomena (rather than simply creating visual simulations – see section 6.3, where Zen and Sean were modifying a descriptive program of a ball falling towards the ground to represent causality of the movement).

To model natural phenomena, students need to see the code as a representation of the mechanism that causes the phenomenon, rather than as a tool for creating a simulation that would simply show how the phenomenon looks. Simulations created from models that are representations of the causal mechanism of the phenomenon, should simply support the models (programs), in being quick ways to show how particular models work, rather than being the focus of students' work. In this study, I have highlighted instances where students started seeing their programs as the models, suggesting that despite their possible disadvantages, SC and MW can be used as modeling tools in science.

7.4.5. Descriptive vs. causal models of natural phenomena

Models of physical phenomena can be *descriptive*, simply showing or describing what the phenomenon looks like. Students' early programs in MW included a number of instructions that resulted, for instance, in an accelerated motion. Instructions however did not include the mechanism that was causing the changes in the speed of the object. Despite the fact that their instructions seemed to have an underlying mechanism that was causing the change in the behavior of the object (e.g., in subsequent lines of code, the speed was increasing in a particular pattern), the actual mechanism was not represented in the program. In a way, students were creating simple *descriptive* models. Students' early programs in SC included a number of subsequent "scenes" of the phenomenon by creating rules that simply assigned different behaviors to objects. In SC, descriptive models of this sort were usually longer than causal models (whereas in MW descriptive programs were usually longer than causal models).

Models can also be *causal*, which are representations of what is causing the phenomenon. A descriptive model is a program that includes a succession of behaviors of an object, whereas a causal model is a program that includes code that represents the mechanism that causes the changes in the object’s behaviors. Thus, a causal model can *(re)create* (instead of just showing) a number of subsequent “scenes.”

In this study, what was missing from students’ descriptive models in SC was a representation of the relation between the different snapshots, such as the rate of the change of the speed. Because of that, one may argue that SC students’ early programs were not scientific models, but rather simple representations of the phenomenon made up of different scenes that follow each other. At the same time, however, this kind of program could provide students with the opportunity to discover the need for “inventing” a concept such as speed or velocity and using it in the program. SC students started creating causal models as soon as they identified a relationship between their subsequent rules and tried to represent it in their programs. Students working with MW started to work towards the creation of a causal model as soon as they started reading and talking about the code as the representation of the phenomenon.

7.5. Implications for developing software packages of CPE for young learners

In this study, I have investigated and reported the different ways that fifth grade students used two different CPEs (MW and SC) for developing representations of natural phenomena. The software packages used in this study were chosen based on their characteristics, mainly representing two different programming paradigms (Kiper, Howard, & Ames, 1997): textual and visual programming. Of course, due to the different interface that MW and SC use, they also have additional differences in the way that programming and debugging is done.

In several cases, findings regarding students' work and conversations were related to characteristics of the software that students used. Below, I summarize the main characteristics of the two CPEs that are possibly related with the observed student use of the CPEs, providing a possible framework for development of new software packages for young learners. In doing so, I follow research (e.g., Druin, *et al*, 1999) in the field that investigates usability issues of software packages and their implications for software development. The discussion below focuses on the different ways that students saw their work with CPEs, on differences related with the object-oriented vs. procedural programming interface, and the difficulties of programming vs. the difficulties of reading a program (program representation).

7.5.1. Write a program vs. write a game

Students working with MW saw their work as formal programming. Their conversations were most of the time technical. During planning 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. During programming, they focused on writing programs without bugs. Students working with MW broke their programs down in segments that were meaningful for the phenomenon they wanted to represent: jumping consisted of two subroutines, one for the boy going up and another for the boy coming down.

Students working with SC saw their work as creating 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. Early programs consisted of a number of independent rules, which were each responding to a "scene" of their game. Students broke down their programs based on the ideas they wanted to show in those scenes (e.g., a ball falling with a different rule for its different speeds). During planning and subsequently trying to develop a game in SC, students seemed to have “quality” criteria that were related to the characteristics of computer games. Those criteria included depiction, scoring rules, and “a purpose for their game”. Students’ program decisions were based on features of their games, and they were not concerned with science as much as they were concerned with whether the behavior of different objects would fit in their game scenario.

Due to students’ focus on the overall story/game, there were cases in which “game” and “science” were in conflict. Of course, there were cases in which a “story approach” for using SC was useful in telling a story of a “causal agent” or a variable in their models (such as acceleration, gravity etc), very similar to developing a program that *is* the cause of the phenomenon represented in the simulation. Students using different CPEs saw their work differently. Seeing it as writing programs, may have been more productive for getting into modeling easily whereas creating games through easier programming, was steering students into the direction of creating “quality” games which are not necessarily “quality” models of natural phenomena.

7.5.2. Object oriented vs. procedural programming

The CPEs that were used in this study were in some aspects fundamentally different. MW is a procedural programming environment that uses textual program language (Papert, 1993). SC is an object oriented programming environment, which uses visual if-then rules (Smith & Cypher, 1999). In this sense, SC can be considered a “non-linear” program environment, as opposed to MW, which can be considered a “linear” program environment, since instructions are written in a one-dimensional pattern.

Object oriented media are based on the idea of enabling the user to directly manipulate objects in the program (Smith & Cypher, 1999). Therefore, code (rules) and variables of each object are physically linked to the character, and represented as such through the media. Writing and reading a character's rules is as simple as double clicking on the character. Rather than writing all their entire code in one window in a one-dimensional fashion, SC stores the rules "behind" each character. Unless otherwise indicated, in a given microworld, multiple copies of a character automatically share the same rules of behavior.

A simulation in SC is a result of the characters' individual actions or changes (Ruder, Brand, & Lewis, 1997) that create system actions or system changes (Colella, Klopfer, & Resnick, 2000). This has an important implication for programming: to program a system in SC, one would expect students to identify and subsequently program the *behaviors of individual objects* which in turn would create the *system's actions and changes*. This might be an advantage, helping young learners to focus on individual objects and their actions, behaviors and interactions. On the other hand, it could place a possible difficulty in abstracting the desired system action and translating it into specific rules assigned to characters. For instance, developing a simulation of how traffic patterns occur in a highway during rush hours is a matter of creating a set of rules for the cars that would create those patterns (Colella, Klopfer, & Resnick, 2000). That the traffic builds up and moves backwards (assuming that the cars move forward) would not be reflected in the code for the individual cars; it is just a desired "side effect" of the car's rules of behavior.

When working with SC, however, students were planning their work thinking about the system, its behaviors and characteristics, and not about the individual systems' objects. In this sense, talking about the system and the system changes was not productive for modeling in science because students needed to translate their ideas about the system into rules for the individual objects' behaviors. This seems to be a *possible paradox*: instead of thinking in terms of individual objects, which one would expect because that would be supported by the way modeling is done in SC, students working with SC were planning for the overall story/system actions and changes.

Students working with MW, however, were planning their work by talking about their program's structure based on the objects' characteristics. Having to write instructions about objects, students were developing programs that consisted of a sequence of instructions of what each object would do.

What do these findings suggest about the characteristics of programming environments for young learners? One of the most important points illustrated from this study is that students may not use a software package in a way its creators intended while developing it. More precisely, object oriented programming in SC was not used by students in a way that would support modeling, at least for most of their work. Even though object-oriented interface is used to make programming easier, and thus may be considered more appropriate for young learners, students using SC were not using it productively. Furthermore, programs that students created with SC were successions of rules representing "scenes" from their story, which SC was running in sequence. There seems to be an additional *possible paradox*: students were using a non-linear program environment to represent successive linear events, instead of e.g., using the environment to represent mechanisms that cause the change in the objects' behaviors.

Another difference between SC and MW is the program language they use. Programming in SC is done by *demonstration (programming by example)* using "drag-and-click" techniques (Smith & Cypher, 1999; Cypher & Smith, 1995). To assign a rule of behavior to a character, the user demonstrates the objects' behavior to the computer by "operating the computer interface" (Smith, Cypher, & Tesler, 2000, p76). "The computer records the user's actions and can then re-execute them later on different inputs" (Smith, Cypher, & Tesler, 2000, p76). All interactions with SC are by direct manipulation, with no need to use a keyboard. The appearance of the microworld, of the objects within the microworld and the appearance of the objects are all under the control of the children through simple drawing tools (Underwood, et al, 1996).

In developing SC, the intention was to create a programming environment that would not require any knowledge of a programming language. What exists is a programming syntax (Smith, 1994). Smith, et al, (2000) claim, that this was an effort to bring the software programming environment closer to the user rather than bring the user closer to the programming environment (by learning the program language of the environment). This was also an effort to eliminate the gap between one's mental representation of what is wanted and the computer's representation (Smith, et al, 2000; Smith, 1994), assuming that the visual nature of the code would make the programs easier to understand (Singh, & Chignell, 1992). Because of their choice of ease, SC's designers sacrificed generalizability and programming power. (Smith, et al, 2000)

The process of programming in MW is more formal and thus possibly more difficult because students have to manipulate a symbolic language to write instructions for the objects in their microworlds, whereas programming in SC is much easier and tangible because students are assigning behaviors to characters by simply demonstrating the desired behavior to the system.

7.5.3. Easier programming vs. easier program representation

However, program representation in MW seemed to be much more meaningful, because students could simply read the code to understand the program, whereas program representation in SC was more abstract. Students in SC had to figure out what each rule was doing by interpreting graphical reminders of the actions of each rule. Thus, reading code in MW was much more frequently observed and it seemed to be related to different tasks by the students. It seems that there is a gap between how programming is done and how it is represented. Smith (1994) refers to this as the programming-by-demonstration problem). Traditionally, in textual CPEs programming is done by typing in the code, and reading the program would be simply reading what had been typed. Graphical rewrite rules in SC serve *only* as graphical reminders of the programs. This leads to a *possible paradox*: creating rules (programming) in SC is supposed to be much easier than writing code in SC, but reading rules is much more difficult.

7.6. Significance and implications of this study

This study is a demonstration of the possible pedagogical value of technology used in science education. It offers possible ways to implement technological tools in conceptual understanding in science education, including presentation of promising possibilities for using CPEs as modeling media in science education. I believe that this study can benefit two different research communities. It can provide feedback information and new ideas to developers of software applications for young children and it can provide science educators with new information that can enhance their understanding of young students as programmers while they develop understanding in science. In this sense there are three major outcomes of this study:

7.6.1. Better understanding of how young learners construct knowledge in the form of models in science education.

Future science educators may use data and results from this study to realize how constructing understanding of the physical world can be thought of as constructing simple models of the physical phenomena and making links among these phenomena. They may also understand that constructing models of the physical world can be an open-ended process where young learners search for a suitable model that simulates real life.

7.6.2. Change in the way computer-based programming tools are used for teaching science education.

This study can provide future researchers and educators with a better understanding of how children use technology tools in the classroom for learning. Future researchers and educators may understand that computer-based tools can enable students to construct powerful models of dynamic systems and use them to make predictions in unknown contexts. They also may come to understand that there are different ways of using different types of CPEs for enhancing scientific visualization for young learners and most importantly may understand the different ways that young learners use different types of technological tools of the kind that I have investigated in this study.

7.6.3. Change in the development of software programming tools for young learners.

Future researchers and developers of modeling software applications for young children may come to understand that computer-based tools can serve young learners as a cognitive tool for modeling and learning. They also may realize that there are specific characteristics and features of applications of this kind that are more appropriate for young learners than other characteristics. In this sense, future researchers may use results from this study to inform the characteristics they build into their CPEs, referring to the characteristics of student thinking and learning in science that are supported by different modeling applications.

8. REFERENCES

Anderson, T., Howe, Chr., and Tolmie, A. (1996) Interaction and mental models of physics phenomena: Evidence form dialogues between learners. In J. Oakhill & A. Garnham (Eds.), *Mental Models in Cognitive Science* (pp. 247-273). UK: Psychology Press.

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.

Bell, Ph., Daris, E., A., and Linn, M., C. (1995). The knowledge integration environment: Theory and design. *Proceedings of Conference of Computer Support for Collaborative Learning*, 1-8.

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

Brown, A., L. (1990). Domain-specific principles affect learning and transfer in children. *Cognitive Science*, 15, pp. 107-134.

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

Constantinou, C., P. (1996). The Cocoa microworld as an environment for modeling physical phenomena. *International Journal of Continuing Education and Life-Long Learning*, 8 (2), 65 - 83.

Creswell, W. J. (1988). *Qualitative inquiry and research design: Choosing among five traditions.* Thousands Oaks, CA: Sage Publications, Inc.

Cypher, A., & Smith, D. (1995). KidSim: End user programming of simulations. In *Proceedings of CHI' 95*, ACM Press, NY, p. 27-34.

De Corte E., Verschaffel, L., Schrooten, H., Olivie, H., & Vansina, A. (1993). A Logo-based tool-kit and computer coach to support the development of general thinking skills. In T. M. Duffy, J. Lowyck, D. H. Joassen (Eds.), *Designing environments for constructive learning* (p. 109-124). NY: Springer-Verlag.

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

diSessa, A., A. (1982). Understanding Aristotelian physics: A study of knowledge-based learning. *Cognitive Science*, 6, 37-75.

diSessa, A., A. (1988). Knowledge in pieces. In G. Forman & P. B. Pufall (Eds.), *Constructivism in the computer age*. Hillsdale, NJ: Lawrence Erlbaum Associates.

diSessa, A., A., Hammer, D., Sherin, Br., & Kolapakowski, T. (1991). Inventing graphing: Meta-representational Expertise in Children. *Journal of Mathematical Behavior*, 10, pp.117-160.

Driver, R. and Oldham, V. (1986) A constructivist approach to curriculum development in science. *Studies in Science Education*, 13, 105-122.

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.

Duschl, R. A. (1990). *Restructuring science education: The importance of theories and their development* (New York: Teachers College Press).

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

Enkenberg, J. (1989). Computer programming, Logo and development of thinking. In G. Schuyten & M. Valcke (Eds.), *Teaching and learning in Logo-based environments: Proceedings of the Eurologo 1989 Conference* (pp. 150-165) Washington DC: IOS.

Fay, S., L., & Mayer, R., M. (1987). Children's naïve conceptions and confusions about Logo graphic commands. *Journal of Educational Psychology*, 79 (3), 254-268.

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

Gelman S., A., & Markman, E., M. (1986). Categories and induction in young children. *Cognition*, 23, pp. 183-208.

Gilbert, J., K. and Boulter, C. (1995). The role of models and modeling in some narratives of science learning. *Paper presented at the Annual Conference of the American Educational Research Association* (San Francisco, CA).

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. (1994). Epistemological beliefs in introductory physics. *Cognition and Instruction*, 12 (2),151-183.

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.

Hestenes, D. (1997) Modeling methodology for physics teachers. In E.F. Redish and J.S. Rigden (Eds.), *The changing role of physics departments in modern universities: Proceedings of International Conference on Undergraduate Physics Education* (p. 935-957). NY: The American Institute of Physics.

Hogan, K., Natasi, B., K., & Pressley, M. (2000). Discourse patterns and collaborative scientific reasoning in peer and teacher-guided discussions. *Cognition and Instruction, 17*(4), 379–432

Johnson-Laird, P. N. (1990). *Mental Models: Towards a cognitive science of language, inference and consciousness*. Cambridge: University Press.

Kahn, K. (1999). Helping children learn hard thinks: Computer programming with familiar objects and actions. In A. Druin (Ed.), *The design of children's technology*. San Francisco: Morgan Kaufmann Publishers, Inc.

Karmiloff-Smith, A., & Inhelder, B. (1974). "If you want to get ahead, get a theory." *Cognition*, 3, pp. 195-212.

Kelly G., J., Druker, S. & Chen, D. (1998). Students' reasoning about electricity: combining performance assessments with argumentation analysis. *International Journal of Science Education*, 20, 849-871.

Kindfield, A., C., H. (1995) Constructing models of biological processes through reasoning and diagrams. *Paper presented at the Annual Conference of the American Educational Research Association* (San Francisco, CA).

Kiper, S., D., Howard, E., & Ames, Ch. (1997). Criteria for evaluation of visual programming languages. *Journal of Visual Languages and Computing*, 8, 175-192.

Krajcik, S., Soloway, E., Blumenfeld, Ph., & Mary, R. (1998). Scaffolded Technology tools to promote teaching and learning in Science. In Ch. Dede (Ed.). *Learning with Technology* (p. 31-45). Alexandria, VA: Association for Supervision & Curriculum Development.

Kuhn, D. (1989). Children and adults as intuitive scientists. *Psychological Review*, 96 (4), pp. 674-689.

Kuhn, D. (2001). How do people know? *Psychological Science*, 12 (1), pp. 1-8.

Kurth, A. L., Kidd, R., Gardner, R., & Smith, Ed. L. (2002). Student use of narrative and paradigmatic forms of talk in elementary science conversations. *Journal of research in science teaching*, 39 (9), 793-818.

Lehrer, R., Lee, M., & Jeong, A. (1999). Reflective teaching of Logo. The *Journal of the Learning Sciences*, 8(2), 245-289.

Löhner, S., & van Joolinger, W. (2002). The effects of representations on communication and product during collaborative modeling. In G. Stahl (Ed.), *Proceedings of Conference of Computer Support for Collaborative Learning (CSCL)* (pp. 463-471). NJ: Lawrence Erlbaum Associates, Inc.

Louca, L. & Constantinou, C. *Using computer-based microworlds for constructing modeling skills in physical science: an example from light*. Manuscript submitted for publication.

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., 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.

Louca, L., Hammer, D., & Bell, M. (2002). Developmental versus context-dependant accounts of abilities for scientific inquiry: A case study of 5-6th grade student inquiry from a discussion about a dropped pendulum. In P. Bell, R. Stevens & T. Satwicz (Eds.), *Keeping Learning Complex: The Proceedings of the Fifth International Conference of the Learning Sciences (ICLS)* (pp. 261-267) Mahwah, NJ: Erlbaum.

Louca, L., Hammer, D., & Kagey, T. (2003). Specificity of epistemological knowledge: Context dependencies of epistemological resources. An example from a science discussion in a 3rd grade. Paper presented in *AERA 2003 conference*, Chicago, IL.

Louca, L., Elby, A., Hammer, D. & Kagey, T. (2003). *Epistemological resources: Applying a new epistemological framework to science instruction*. Manuscript accepted for publication.

Medawar, P. (1987). *Pluto's Republic*. Oxford: Oxford University Press.

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

Metz, K. (1995). Reassessment of developmental constraints on children's science instruction. Review of *Educational Research*, 65 (2), pp. 93-127.

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

Orhun, E. (1993). Learning problem solving through computer programming. In D. L. Ferguson (Ed.). *Advanced educational technologies for mathematics and science* (p. 339-362). NY: Springer-Verlag.

Papert, S. (1980). *Mindstorms. Children, Computers & Powerful Ideas*. NY: Basic Books, Inc. Publishers.

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.

Pea, R., D., Kurland, M. D., & Hawkins, J. (1987) Logo and the development of thinking skills. In R. D. Pea & K. Sheingold (Eds.). *Mirrors of minds: Patterns and experience in educational computing* (p. 178-197). Norwood, N.J.: Ablex Publishing Corporation.

Penner, D. (2001). Cognition, Computers and synthetic science: Building knowledge and meaning through modeling. In Walter G. Secada (Ed.). *Review of Research in Education*. AERA: Washington, DC.

Penner, D., E., Giles, N., D., Lehrer, R., & Schauble, L. (1997). Building functional Models: Designing an Elbow. *Journal of Research in Science Teaching*, 34 (2), pp. 125-143.

Penner, D., Lehrer, R., Schauble, L. (1998). From physical models to biomechanics: A design-based modeling approach. *The Journal of the Learning Sciences*, 7(3&4), 429-449.

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

Raghavan, K. and Glaser, R. (1995) Model-based analysis and reasoning in science: The MARS curriculum. *Science Education*, 79 (1), 37-61.

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.

Rochelle, J. (1992). Learning by collaborating: convergent conceptual change. *The Journal of the Learning Sciences*, 2, 235-276.

Samarapungavan, A. (1992). Children’s judgments in theory choice tasks: Scientific rationality in childhood. *Cognition*, 45, p. 1-32.

Schecker, H., P. (1993). The didactic potential of computer aided modeling for physics education. In D.L. Ferguson (Ed.). *Advanced Educational Technologies for Mathematics and Science*. NY: Springer-Verlag (NATO series)

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.

Singh, G., & Chignell, M., H. (1992). Components of the visual computer. *The Visual Computer*, 9, 115-142.

Singh, J., K. (1992). Cognitive effects of programming in Logo: A review of literature and synthesis of strategies for research. *Journal of Research in Computing in Education*, 25(1), 88-104.

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.

Thompson, P., W. (1985, September). A Piagetian approach to transformation geometry via microworlds. *Mathematics Teacher*, 465-471.

Toulmin, S. (1958). *The Uses of Argument.* Cambridge: Cambridge University Press).

Underwood, G., Underwood, J., Pheasey, K., & Gilmore, D. (1996). Collaboration and discourse while programming the KidSim microworld simulation. *Computers & Education*, 26, 143-151.

van Zee, E., (2000), Analysis of a student-generated inquiry discussion. *International Journal of Science Education*, 22 (2), 115-142.

van Zee, E., and Minstrell, J. (1997) Reflective discourse: developing shared understandings in a physics classroom. *International Journal of Science Education*, 19, 209- 228.

van Zee, E., Iwasyk, M., Kurose, A., Simpson, D., & Wild, J. (2001). Student and teacher questioning during conversations about science. *Journal of research in science teaching*, 38 (2), 159-109.

Verschsffel, L., De Corte, E., Schrooten, H., Ondemans, R. & Hoedemaekers, E. (1989). Cognitive effects of programming and instruction in sixth grades. In G. Schuyten & M. Valcke (Eds.), *Teaching and learning in Logo-based environments: Proceedings of the Eurologo 1989 Conference* (pp. 150-165). Washington DC: IOS.

Vosniadou, S. (1989). Analogical reasoning and knowledge acquisition. In S. Vosniadou & A. Ortony (Eds.) *Similarity and analogical reasoning* (p. 413-437). Cambridge: Cambridge University Press.

Wells, M, Hestenes, D. and Swackhamer, G. (1995) A modeling method for high school physics instruction. *American Journal of Physics*, 63 (7), 606-619.

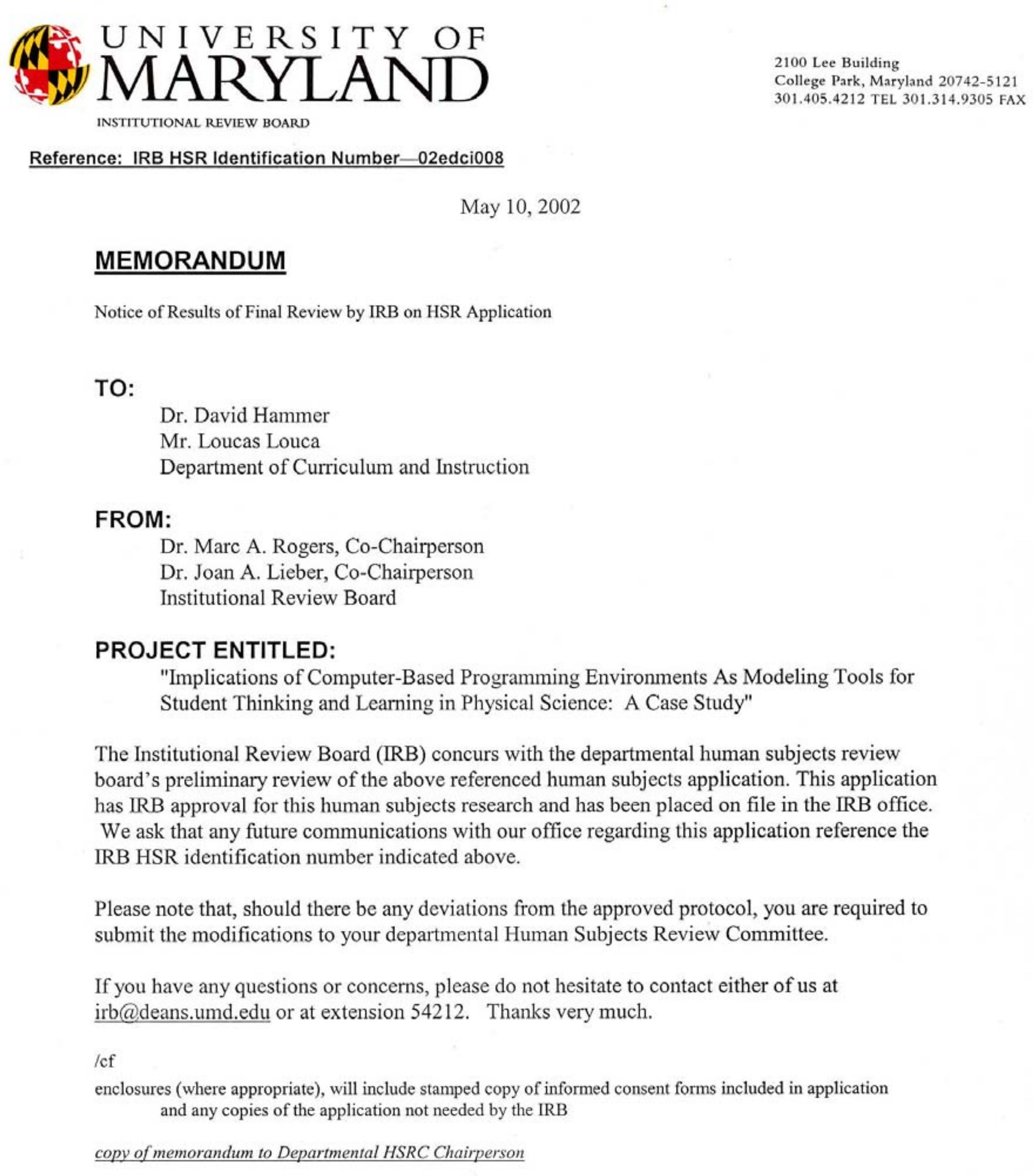
White, B. Y. and Frederiksen, J. R. (1998). Inquiry, modeling and metacognition: Making science accessible to all students. *Cognition and Instruction*, 16 (11), 3-118.

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.

Yin, K. R. (1994). *Case study research: design and methods*. Thousands Oaks, CA: Sage Publications, Inc.

9. APPENDIX

9.1. IRB approval for the study



UNIVERSITY OF MARYLAND

INSTITUTIONAL REVIEW BOARD

2100 Lee Building
College Park, Maryland 20742-5121
301.405.4212 TEL 301.314.9305 FAX

Reference: IRB HSR Identification Number—02edci008

May 10, 2002

MEMORANDUM

Notice of Results of Final Review by IRB on HSR Application

TO:

Dr. David Hammer
Mr. Loucas Louca
Department of Curriculum and Instruction

FROM:

Dr. Marc A. Rogers, Co-Chairperson
Dr. Joan A. Lieber, Co-Chairperson
Institutional Review Board

PROJECT ENTITLED:

"Implications of Computer-Based Programming Environments As Modeling Tools for Student Thinking and Learning in Physical Science: A Case Study"

The Institutional Review Board (IRB) concurs with the departmental human subjects review board's preliminary review of the above referenced human subjects application. This application has IRB approval for this human subjects research and has been placed on file in the IRB office. We ask that any future communications with our office regarding this application reference the IRB HSR identification number indicated above.

Please note that, should there be any deviations from the approved protocol, you are required to submit the modifications to your departmental Human Subjects Review Committee.

If you have any questions or concerns, please do not hesitate to contact either of us at irb@deans.umd.edu or at extension 54212. Thanks very much.

/cf

enclosures (where appropriate), will include stamped copy of informed consent forms included in application
and any copies of the application not needed by the IRB

copy of memorandum to Departmental HSRC Chairperson