Fostering Synergistic Learning of Computational Thinking and Middle School Science in Computer-based Intelligent Learning Environments

By

Satabdi Basu

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

May, 2016

Nashville, Tennessee

Approved:

ACKNOWLEDGEMENTS

This dissertation would not have been possible without the constant guidance, support, and encouragement of a number of people. Family, friends, my Ph.D. advisor, my dissertation committee, co-workers, and funding agencies all share in the credit.

First of all, I would like to thank my advisor, Professor Gautam Biswas, for giving me the opportunity to work with him, and being a continuous source of inspiration, support, encouragement, and constructive criticism. Next, I would like to thank my dissertation committee: Professor Julie A. Adams, Professor Douglas Fisher, Professor Douglas Clark, and Professor Pratim Sengupta for their time, advice, guidance, suggestions, stimulating discussions, and words of encouragement. I am grateful to the National Science Foundation for financially supporting this dissertation research under their cyber-learning grants 1124175 and 1441542.

I would like to thank the teachers, students, and administration at the middle schools where I conducted my research studies for this dissertation. In particular, I would like to thank Ms. Jamie Schimenti and Mr. James Parsons for allowing me to conduct research studies in their classrooms and for their time, co-operation, support, and useful suggestions.

During my graduate study at Vanderbilt University, I have also been fortunate to have friends, co-workers and research colleagues who have enriched my graduate experience in different ways. I would like to thank current and past members of my research project for their collaboration, numerous fruitful discussions, and useful suggestions. I would also like to thank friends at Vanderbilt University and beyond for being supportive and patient over the years, providing honest opinions about my research, being my tireless cheering squad, and ensuring I learn to balance my dissertation research with other aspects of my life.

Finally, I will be forever grateful to my family for their unconditional love and support throughout my life. I would like to thank my parents, Sukla Basu and Udayan K. Basu, for their unwavering support and encouragement, and being my most tireless advisors and cheerleaders from halfway across the world. Thank-you for believing in me, and teaching me to believe in myself! Last, but certainly not the least, I would like to thank my husband, Tamoghna Bhattacharya, for being a source of constant support and patiently seeing me through the daily ups and downs of my Ph.D. journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CTSiM v1	Computational Thinking using Simulation and Modeling version 1
CTSiM v2	Computational Thinking using Simulation and Modeling version 2
CT	Computational Thinking
STEM	Science, Technology, Engineering and Mathematics
NRC	National Research Council
ISTE	International Society for Technology in Education
CSTA	Computer Science Teachers Association
ACM	Association for Computing Machinery
CoLPACT	Context, modeling Language, Pedagogy, and Assessments for Computational Thinking
ABM	Agent Based Modeling
MABM	Multi Agent Based Modeling
NETS	National Educational Technology Standards
ITS	Intelligent Tutoring System
PBS	Program Behavior Similarity
CTP	Computational Thinking Pattern
ESW	Eco Science Works
GUTS	Growing Up Thinking Scientifically
IPRO	I can PROgram
TNG	The Next Generation
NIELS	NetLogo Investigations in Electromegnetism
MAGG	Measuring AGGregation
VP	Visual Programming
DSML	Domain Specific Modeling Language
C-World	Construction World
E-World	Enactment World
V-World	Envisionment World
S-Group	Scaffolded Group
C-Group	Classroom Group
TCAP	Tennessee Comprehensive Assessment Programme

RC	Roller-Coaster
MER	Multiple External Representations
OELE	Open Ended Learning Environment
IA	Information Acquisition
SC	Solution Construction
SA	Solution Assessment

CHAPTER 1

Introduction

Recent advances in computing are transforming our lives at an astonishing pace by unleashing the potential of technology to solve various pressing global problems. This has allowed researchers in all disciplines to envision new and innovative problem-solving strategies, which can be systematically tested and evaluated virtually before actual deployment. This influence of computing naturally gives an upper hand to those who learn to utilize computational methods and tools to understand, formulate, and solve problems in different disciplines over those who lack such skills. Computational Thinking (CT) is a term used to describe a broad range of mental processes fundamental to computer science that help people find effective methods to solve problems, design systems, understand human behavior, and engage the power of computing to automate a wide range of intellectual processes (Wing, 2006; NRC, 2010). Driven by the needs of a 21st century workforce, CT now routinely features as an essential element of K-12 curricula worldwide (ISTE, 2007; UKEd13), and there is a great emphasis on teaching students to think computationally from an early age.

In order to make CT accessible to all students and to successfully embed CT into the K-12 curricula, it needs to be integrated with existing K-12 curricula or introduced as new curricular material. Science, Technology, Engineering, and Mathematics (STEM) disciplines lend themselves particularly well to integration with CT. CT is considered a vital ingredient of STEM learning (Wing, 2006; Barr & Stephenson, 2011; Grover & Pea, 2013; Sengupta et. al., 2013; Jona et. al., 2014), and has been included as a key feature in K-12 science education frameworks (NRC, 2011). However, efforts to integrate CT with science learning or learning in other STEM disciplines have been limited, especially at the elementary and middle school level. Also, curricular integration of CT requires development of systematic CT assessments, an area that is under-investigated despite its importance being well recognized (ACM and CSTA report, 2011; Grover & Pea, 2013). Similarly, there is dearth of research studying students' learning and developmental processes while learning CT skills and using CT-based learning environments, and developing scaffolds for CT learning.

It is within this context that the present dissertation research is situated. In particular, this research provides a novel approach for the systematic design and implementation of a CT-based science learning environment for middle school students, which can be integrated into the middle school curricula by teachers with little prior programming experience. We also develop a set of assessments to measure students' learning of science and CT content in this environment. In the last phase of our work, we use some of the assessment metrics online to interpret and analyze student behavior and performance, and then adaptively scaffold students to help them become better learners.

1.1 Problem overview: Integrating CT into middle school science curricula

Wing (2006) promoted the term 'Computational Thinking' and voiced that CT "represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use." CT skills include reformulating seemingly difficult problems into solvable forms using reduction, transformation, recursion, and simulation, choosing appropriate representations for problems, and modeling relevant aspects (and ignoring irrelevant aspects) of problems to make them tractable. These skills also provide features that support using different levels of abstraction for problem solving, and decomposing large complex tasks into manageable modular subtasks that supports parallel execution and multiple problem solvers (Wing, 2008).

Wing argued that CT should be included as a determinant of every child's analytical ability along with reading, writing, and arithmetic by the middle of the 21st century. Just like young students initially learn to read so that they can later read to learn, they also need to learn to think computationally so they might later use it to learn complex concepts, represent solutions as computational steps, and solve problems using computational models and methods. Developing CT skills is also believed to lead to increased abilities to deal with open-ended and complex problems, and to communicate and collaborate to achieve common goals (CSTA, 2011). With the proliferation of computing concepts and computational devices, it is no longer sufficient to wait until students are in college to introduce them to CT concepts. Students must begin to work with algorithmic problem solving and computational methods and tools during their K-12 years, preferably starting at the middle school level (Barr & Stephenson, 2011).

An approach to addressing the absence of CT concepts in standard middle school curricula can be seamlessly integrating CT with existing components of the middle school curricula. CT skills lend themselves particularly well to STEM learning, since several of the epistemic and representational practices central to the development of expertise in STEM disciplines (e.g., characterizing problems and designing solutions, developing and using models, analyzing and interpreting data) are also primary components of CT (NRC, 2011). With decades of use of computational modeling and simulation, and today's use of data mining and machine learning algorithms to analyze massive amounts of data, computation is recognized as the third pillar of science, along with theory and experimentation (PITAC, 2005). Computational modeling and abstraction not only parallel core practices of science education, but can also serve as effective vehicles for learning challenging science concepts, and modeling and analyzing complex, multi-dimensional scientific models (Guzdial, 1995; Sherin, 2001; Kynigos, 2007; Hambrusch et al., 2009; Blikstein & Wilensky, 2009).

While the synergy between CT and science learning is well recognized, leveraging the synergy successfully in middle school classrooms is a challenging task. Leveraging the synergy involves choosing topics in the existing science curriculum, and then developing newer approaches to learning and problem solving that seamlessly incorporate the use of CT skills, while ensuring minimal overhead for students and teachers in terms of time, effort, training and other resources. Further, the introduction of CT into the curriculum drives the need for developing systematic assessments for CT. Such assessments can provide a thorough understanding of students' difficulties in using computational methods and tools, which can then lead to the development of systematic scaffolds to support students' development and use of CT skills. These areas are currently under-investigated and under-developed, especially at the middle school level.

Therefore, it is not surprising that efforts to integrate CT skills with science learning have been limited, especially at the elementary and middle school levels. Today, several CT-based learning environments used by middle school students (graphical programming environments, such as Scratch, Alice, Game Maker, Kodu and Greenfoot; and robotic kits and tangible media, such as Arduino and Gogo Boards - Grover and Pea, 2013) do little to leverage the synergy between CT and science. They engage students through motivating contexts like game-design, story-telling, and app-design, and generally find use in after-school workshops, summer camps, or as part of other extra-curricular activities where the primary focus is increasing students' interest in and engagement with computational tools, but not on learning of science or other STEM concepts. There are some exceptions like AgentSheets (Repenning, 2000), EcoScienceWorks (Allan et. al., 2010), CT-STEM (Jona et. al., 2014), and Project GUTS (http://projectguts.org/), which focus on integrating CT with middle school curricular topics through computational modeling of science topics. But, even in these environments, assessments for measuring learning of CT concepts and skills are lacking, as are scaffolds for helping students with their modeling tasks. Some efforts have also been made for integrating CT with STEM learning in high school and undergraduate classrooms, and they have generally involved exposing students to a general purpose high level programming language like Python or Scheme for a few weeks before giving students assignments to model scientific phenomena using the programming language (Hambrusch, et. al, 2009, Google's exploring Computational Thinking website). Unsurprisingly, such approaches have not been adopted in K-12 classrooms with younger children – learning languages like Python creates a large overhead for both young students and their instructors, who have had no prior programming experience, and spending substantial chunks of time on learning these languages is not feasible given curriculum constraints.

Integrating CT with middle school STEM curricula is, therefore an important but non-trivial task. Promoting CT in extra-curricular activities or elective classes cannot be a long-term solution, since it makes CT accessible to only a selected few. While curricular integration with science topics may be a particularly effective way to introduce CT concepts and practices into middle school curricula, we also need to ensure that the approach is manageable and adds pedagogical value for teachers and students by creating a framework for synergistic and simultaneous science and CT learning. In addition, to support teachers and demonstrate pedagogical value, standardized CT assessments need to be developed, and teachers need to be made aware of the potential challenges faced by students in the combined CT and science learning curriculum. In this dissertation research, we address these limitations by developing a sequence of curricular units that demonstrate the synergistic learning of science and CT concepts as part of middle school science curricula. To facilitate learning of CT and science content and to support the classroom science teachers, we have developed a computer-based learning environment that combines a visual programming language for computational modeling of science phenomena with simulation and visualization tools that help students study science processes through model building, testing, analysis and verification. Furthermore, we have developed standardized, objective, and holistic assessments to study student competency in CT concepts and practices that are supported in this learning environment, and developed adaptive scaffolding mechanisms to help students overcome difficulties they have with their learning and problem solving tasks in this environment.

1.2 Research Approach and Contributions

This dissertation research evolved in two primary phases of work, each of which has produced a number of research contributions.

The first phase of research involved the design, development, and evaluation of an initial version of the computer-based Computational Thinking using Simulation and modeling (CTSiM) learning environment for combined CT and science learning in middle school classrooms. The initial design and implementation of CTSiM, and the design and execution of a research study using the initial version of CTSiM were both performed collaboratively by a research team that included Computer Science and Peabody College of Education researchers at Vanderbilt University (Basu et. al., 2012; Sengupta et. al., 2013; Basu et. al., 2013). Students used CTSiM to learn by constructing, evaluating, and revising computational models of science processes in two domains – Kinematics and Ecology.

Building computational models of science phenomena is, however, a complex and challenging task for middle school students who may not be well-versed in the science topic and have little experience in applying the abstract thinking processes associated with constructing computational models. In addition, students may be unaware of practices that encompass debugging and verification of models. The primary goal of the first research study was to understand the difficulties students faced when simultaneously learning science and CT concepts, and how to support these learning processes and help students develop effective learning behaviors. The study was conducted as a think-aloud study, where researchers worked one-on-one with students building simulation models in CTSiM. The researchers provided verbal scaffolds to help students overcome difficulties that impeded their learning processes. Results from the study revealed that the intervention involving computational modeling using CTSiM helped students learn science concepts, but students faced a number of difficulties in understanding and applying domain knowledge and CT constructs as well as systematically debugging their models, and required various scaffolds to help overcome their difficulties (Basu et. al., 2013).

Lessons learned from the first phase motivated the second phase of our research, where we made modifications and improvements to the CTSiM environment based on students' difficulties that we documented in the first phase of the research. We developed a new version of CTSiM that involved the design and implementation of new CTSiM interfaces, and modifications to improve some developed in Phase 1. Additional tools and functionalities were developed to scaffold student learning (Basu et. al., 2015; Basu, Biswas & Kinnebrew, 2016). Some of the scaffolds were made available to students at all times, while others were provided adaptively based on students' actions in the CTSiM environment. The effectiveness of our adaptive scaffolding framework was evaluated through a controlled classroom study with ninety-eight 6th grade students from a Tennessee middle school. We defined a set of online and offline measures to assess students' science and CT learning, and characterize their learning behaviors and use of CT practices, and used these measures to establish the effectiveness of our adaptive scaffolding framework.

In summary, the proposed research contributes to advancing research in the field of CT-based learning environments in the following ways:

1. A theoretical understanding of the different facets involved in designing synergistic CT and science learning environments and supporting curricula, along with a novel framework for analyzing CT-based environments in terms of these facets.
2. A theoretically grounded justification for the design and implementation of a computer-based learning environment that promotes simultaneous learning of CT and middle school science.
3. The use of empirical think-aloud studies to discover challenges faced by students working in CT-based science learning environments. Understanding these challenges helped inform development of supporting tools, adaptive scaffolds, and assessments for such learning environments.
4. The development of a novel approach for quantifying and assessing students' science and CT learning and use of CT skills in CT-based science learning environments using multiple

assessment modes, some external to the system and some based on measures derived from students' actions in the system.

5. The development and evaluation of an adaptive scaffolding strategy for CT-based science learning environments.

1.3 Organization of the rest of the dissertation

The remainder of this dissertation is organized as follows. Chapter 2 reviews the field of Computational Thinking and related literature on existing CT-based learning environments and curricula. It highlights under-investigated areas in the field, which, in turn, motivates this dissertation research. Chapter 3 presents a theoretical grounding for the initial design and architecture of the CTSiM learning environment, along with a progression of learning activities designed to align with middle school curricular requirements in science. Chapter 4 details a research study conducted with the initial version of CTSiM, and identifies and analyzes the different categories of difficulties students faced in the different learning activities, and the types of scaffolds which helped overcome the difficulties. Based on observed student difficulties described in Chapter 4, Chapter 5 discusses the changes made to the CTSiM architecture and interfaces, while Chapter 6 presents the generalized adaptive scaffolding framework adopted in CTSiM, the process of modeling learners based on their modeling performances and learning behaviors, and the principles governing delivery of feedback via a pedagogical agent in the system. Chapter 7 describes the classroom study we conducted with a version of the system that included adaptive scaffolding. The results establish the effectiveness of the adaptive scaffolding framework. This is demonstrated through different forms of science and CT assessments, some external to the system and some based on an analysis of students' actions logged by the system. Chapter 8 summarizes the contributions of this research, and discusses future research directions.

CHAPTER 2

Background review: Computational Thinking in K-12 Curricula

This chapter presents operational definitions of CT and describe the advantages of integrating CT and science learning in the K-12 curricula. We also present a novel framework for describing and analyzing CT-based learning environments and use this framework to review existing efforts to introduce CT to K-12 students.

2.1 Computational Thinking: Definition and role

When the term 'Computational Thinking' was coined in 2006, it was used to represent a universally applicable set of mental processes fundamental to computer science that involve leveraging the power of computing to solve problems, design systems, and understand human behavior (Wing, 2006). According to Wing (2008), the "nuts and bolts" of CT involve defining multiple layers of abstraction, understanding the relationships between the layers, and deciding which details need to be highlighted (and complementarily, which details can be ignored) in each layer when trying to understand, explain, and solve problems in a particular domain. Computing processes, whether executed by humans or by machines or by a combination of humans and machines, help automate these abstractions and the relationships between the abstraction layers (Wing, 2008). Recently, Aho (2012) simplified the original definition of CT to thought processes involved in formulating problems so that "*their solutions can be represented as computational steps and algorithms*".

While most existing definitions of CT describe it as a 'thought process', Hemmendinger (2010) suggested that "we talk less about computational *thinking*, and focus more on computational *doing*s". Sengupta et. al. (2013) also posited that "*CT becomes evident only in particular forms of epistemic and representational practices that involve the generation and use of external representations (i.e., representations that are external to the mind) by computational scientists*". This pedagogical perspective is important since it means that engaging students in the process of developing abstractions and engaging in other computational representational practices is required in order to support the development of their CT skills. This perspective also aligns with the 'learning-by-design pedagogy', which suggests that students learn best when they engage in the design and consequential use of external representations for modeling and reasoning (Papert, 1991; Kolodner et al., 2003; Blikstein and Wilensky, 2009).

Currently, there are a number of operational definitions of CT used by researchers to describe or analyze their work in this area. For example, several researchers refer to the definition of CT offered by the Carnegie Mellon Center for Computational Thinking which identifies the 3 important aspects of CT as: (1) thinking algorithmically, (2) making effective use of abstraction and modeling, and (3) considering and understanding scale (http://www.cs.cmu.edu/~CompThink/). In 2009, the Computer Science Teachers Association (CSTA) and the International Society for Technology in Education (ISTE) developed an operational definition of CT for K-12, identifying the core CT concepts and capabilities as data collection, data analysis, data representation, problem decomposition, abstraction, algorithms and procedures, automation, parallelization, and simulation (Barr & Stephenson, 2011). Grover and Pea (2013) expanded this list of core CT elements by adding algorithmic notions of flow of control; iterative and recursive thinking; conditional logic; efficiency and performance constraints; and debugging and systematic error detection.

Recently, Brennan and Resnick (2012) described CT as a three-dimensional framework comprising computational concepts, practices, and perspectives. Computational concepts refer to elements, such as sequences, loops, parallelism, events, conditionals, operators, and data structures that are present in many programming languages. Computational practices refer to activities, such as being incremental, reusing and remixing, testing and debugging, and modularizing and abstracting that designers use to create programs. Computational perspectives, such as expressing, connecting, questioning, potential study and career path in computing, and personal relevancy of computing refer to worldviews that designers develop as they engage with digital media, and how they see themselves within the field and the realm of future careers. In this dissertation research, we adopt the three-dimensional CT framework (Brennan and Resnick, 2012) to describe CT skills fostered by our learning environment and compare them against those promoted in other CT-based environments.

It is noteworthy that though CT is defined to draw on concepts fundamental to computer science, it is distinct from computer science in that it only involves seeking algorithmic approaches to problem solving. There is an important distinction between CT and traditional programming in that CT focuses on conceptualization and developing ideas on how to solve a problem rather than dealing with the rigid syntax of programming languages for producing artifacts that represent the solution to the problem. Another question that inevitably arises while describing what CT involves is: How and to what extent are computers essential for developing and working with CT skills? Since CT skills involve representing, simplifying and solving problems, can CT be taught divorced from the use of computers? Though some CT concepts and principles can be introduced and explored without the use of computers, as is done in programs like CS Unplugged (http://csunplugged.org/), such an approach deprives learners of crucial computational experiences (NRC, 2010). In other words, computers and other computational devices may not be synonymous with CT, but they are enablers of CT.

Also, several CT skills might be synonymous with concepts fundamental to computer science, but they are definitely not exclusive to the field. Hemmendinger (2010) points out that the elements of CT like constructing models, finding and correcting errors, and creating representations are shared across multiple disciplines. For example, abstractions are used in all disciplines where modeling is a key enabler for conceptualization and problem solving, such as in science, engineering, mathematics, and economics. Furthermore, reformulating hard problems and separation of concerns are typical of all domains of problem solving. Also, CT is often compared with other forms of thinking, since it shares common features with many of them. For example, algorithmic thinking involves a detail-oriented way of thinking about how to accomplish a particular task or solve a particular problem. CT surely involves algorithmic thinking, but CT also encompasses the representation and interpretation of data, with algorithms providing the tools for analysis and interpretation (Guzdial, 2010). Similarly, CT is clearly related to, but not identical with, mathematical thinking. Both CT and mathematical thinking have an underlying linguistic structure for precisely describing how to do things (algorithms), and both use abstraction and reasoning with simplified models to solve problems. But, mathematical thinking is more about abstract structure than abstract methodology (NRC, 2010). In CT, the layers of abstractions are tightly coupled such that their generation and analysis can be automated. CT also parallels engineering problem solving in several ways since they both deal with design, constraints, modifiability, scalability, cost, performance, and efficiency. However, engineering thinking requires accounting for errors and computing tolerance levels, whereas CT allows building virtual worlds that can be unconstrained by physical reality (Wing, 2006). Procedural thinking is also emphasized in CT, though CT further involves declarative models. Procedural thinking includes developing, representing, testing, and debugging procedures, and an effective procedure is a detailed step-by-step set of instructions that can be mechanically interpreted and carried out by a specified agent, such as a computer or automated equipment.

Today, the wide spectrum of CT applications encompasses disciplines as diverse as science, mathematics, music, poetry, archaeology, and law (NRC, 2011). CT skills are believed to support inquiry and working with complex open-ended problems (CSTA, 2011), and most disciplines involve problem solving, information retrieval and representation, modeling, debugging, testing, and efficiency considerations in some form or the other. Further, CT provides a basis for developing powerful representational practices and can help create representational shifts, producing better modelers among students, and enabling wider access to models across several domains (NRC, 2011). In summary, we see that the ideas behind CT have existed for several years and shares elements with several existing forms of thinking discussed in the literature. However, features like abstraction, automation, algorithm development, modeling and simulation, and making sense of data are some of CT's core ideas, distinguishing it from other types of thinking (NRC, 2010). The ultimate goal of CT should, therefore, not be to teach everyone to think like a computer scientist, but rather to teach them to apply the common set of core elements to solve problems within and across a wide variety of disciplines (Hemmendinger, 2010). In the next section, we discuss the synergy between CT and science learning and the pedagogical benefits and implications of integrating the two.

2.2 Integrating CT and science learning

CT is considered to be at the core of all STEM disciplines (Henderson, Cortina, & Wing, 2007), and several researchers have shown that programming and computational modeling can serve as effective vehicles for learning challenging science concepts (Guzdial, 1995; Sherin, 2001; Kynigos, 2007; Hambrusch et al., 2009; Blikstein & Wilensky, 2009). Complementarily, Harel and Papert (1991) argued that programming is reflexive with other domains, i.e., learning programming in concert with concepts from another domain can be easier than learning each separately. Along similar lines, the ACM K-12 Taskforce (2003) also recommends integrating programming and computational methods with curricular domains, such as science, rather than teaching programming as a separate topic at the K-12 levels. This notion of using computing as a medium for teaching other subjects is not new either. In the context of K–12 education, ‘computing as a medium for exploring STEM domains’ was popularized by Papert (1980). Papert pioneered the idea of children developing procedural thinking and learning about fractions through LOGO programming (Papert, 1980, 1991). Similarly, Guzdial (1994) developed Emile - a scaffolded graphical programming interface to help students learn Physics – and stated that its goal was not to make students learn about programming, but to have students learn through programming since “*programming is a good lever for understanding many domains.*”

Leveraging the synergy between CT and K-12 science requires engaging students in scientific inquiry that involves computational representational practices like defining abstractions, decomposing complex problems, and debugging or systematic error detection. At the broadest level, the process of scientific inquiry involves the generalizable practice of generating models, which are abstractions or generalizable mathematical and formal representations of scientific phenomena (Sengupta et. al., 2013). Similarly, evidence based explanations of any scientific phenomenon involve the generalizable practices of development of hypotheses from theories or models and testing these against evidence derived from observations and experiments (Lehrer and Schauble 2006). Modeling - i.e., the collective action of developing, testing and refining models (NRC, 2008) - involves carefully selecting aspects of the phenomenon to be modeled, identifying relevant variables, developing formal representations, and verifying and validating these representations with the putative phenomenon (Penner et al. 1998; Lehrer and Schauble 2006; Sengupta and Farris 2012). Developing a computational model of a physical phenomenon, therefore, involves key aspects of CT: identifying appropriate abstractions (e.g., underlying mathematical rules or computational methods that govern the behavior of relevant entities or objects), making iterative comparisons of the generated representations and explanations with observations of the target phenomenon, and debugging the abstractions to generate progressively sophisticated explanations of the phenomenon to be modeled.

Therefore, integrating CT and scientific modeling can be beneficial in a number of important ways as listed below (Sengupta et al., 2013):

A. Lowering the learning threshold for science concepts by reorganizing them around intuitive computational mechanisms: Sherin (2001) and diSessa (2000) argued that particular forms of programming could enable novice learners to reason with their intuitions about the physical world. Redish and Wilson (1993) argued that computational representations enable us to introduce discrete and qualitative forms of the fundamental laws, which can be much simpler to explain, understand, and apply, compared to the continuous forms traditionally presented in equation-based instruction. Furthermore, studies also suggest that in the domains of physics and biology, rather than organizing scientific phenomena in terms of abstract mathematical principles, the phenomena can be organized in a more intuitive fashion around computational mechanisms and principles (Redish and Wilson 1993; Sengupta and Wilensky 2011; Wilensky and Reisman 2006).

B. Programming and computational modeling as representations of core scientific practices: Soloway (1993) argued that learning to program amounts to learning how to construct mechanisms and explanations. Therefore, the ability to build computational models by programming matches core scientific practices that include model building and verification, as pointed out earlier.

C. Contextualized representations make it easier to learn programming: When computational mechanisms are anchored in real-world problem contexts, programming and computational modeling become easier to learn. Hambrusch et al. (2009) found that introducing computer programming to undergraduate students who were non-CS majors, in the context of modeling phenomena in their respective domains (physics and chemistry) resulted in higher learning gains (in programming), as well as a higher level of engagement in the task domain.

However, integrating computational modelling and programming with K-12 science curricula can be challenging for a number of reasons. The integration requires development of a sustained and systematic learning progression, which encompasses CT concepts and practices of varying complexities across different science disciplines. Also, it can lead to a high teaching overhead for existing science teachers with no programming experience. Complementarily, learning a programming language and then using it to model a science topic can be challenging and time consuming for students. Therefore, the design of programming-based learning environments needs to be rethought for seamless integration with science education (Guzdial 1995; diSessa 2000; Sengupta 2011). Integrating CT with science in a manner that supports the development of students' scientific expertise requires the design of coherent curricula in which CT, programming, and modeling are not taught as separate topics, but are interwoven with learning in the science domains (Sengupta et. al., 2013). With the recent resurgence in CT based research and the policy attention to STEM learning, there has been an escalated interest in finding ways to tap into the synergy between CT and science education starting at the K-12 level. In the next section, we review efforts to introduce K-12 students to CT concepts and practices, and contrast environments that leverage the synergy between CT and science learning against others that do not.

2.3 CT-based learning environments for K-12 students

With the recognition of the importance of teaching students to think computationally from an early age, strategies for embedding CT across the K-12 curriculum have been proposed, and methods for building CT tools that can be embedded in the K-12 curricula are being recommended by a number of researchers. For example, Repenning, et. al. (2010) present a list of conditions which any CT tool needs to fulfil in order to be successfully integrated into K-12 classrooms: *1) has low threshold* or allows students to produce simple working models quickly*; 2) has high ceiling* or allows students to build highly sophisticated models*; 3) scaffolds flow of learning* or includes a curriculum that gradually increases complexity to manage skills and challenges associated with the tool*; 4) enables transfer* to subsequent computer science applications *; 5) supports equity* or ensures accessibility and motivation across gender and ethnicity boundaries*; and 6) is systemic and sustainable for all teachers and students* meaning the tool should support teacher training and align with curricular standards. Also, the Computer Science Teachers Association (CSTA) and the International Society for Technology in Education (ISTE) identify strategic areas to be addressed for integrating CT with K-12 STEM subject areas. These areas includes helping policy makers connect CT to existing learning goals and standards, providing professional development to K-12 teachers and encouraging them to change courses and curricula by providing them with models and simulations, modeling activities and web sites to support such changes (Barr & Stephenson, 2011). However, in reality, the process of embedding CT in the K-12 curricula has not progressed much, and it leaves a lot to be desired in terms of construction of learning environments which combine CT with STEM learning and existing curricular standards, and development of standardized formative and summative assessments for CT.

In the following sections, we review existing efforts to introduce K-12 students to CT concepts, practices and perspectives. Our literature review shows that only a few of the currently implemented environments attempt to leverage the synergy between CT and science learning. There is also a dearth of systematic CT assessments. To continue our review, we present a comprehensive framework comprising the primary factors that need to be considered when designing a CT-based learning environment or curricula. We then apply this framework to review and analyze existing environments that foster CT skills at the K-12 level. We look at systems that have been developed for formal and informal settings.

2.3.1 CoLPACT – A framework for designing and analyzing CT-based learning environments

Fox (NRC, 2010) proposed that building a CT-based learning environment was a three-dimensional problem with (1) aspects of CT, (2) the disciplines CT is connected with, and (3) pedagogy constituting the three dimensions. In this dissertation research, we extend the Fox framework, and argue that building a CT-based learning environment is a five-dimensional problem with the two added dimensions being (4) the computational language used, and (5) the CT assessments employed in the environment. Taking all five dimensions into account we have developed a novel framework (CoLPACT – **Co**ntext, Computational **L**anguage, **P**edagogy, and **A**ssessments for **C**omputational **T**hinking) that represents the primary dimensions one has to consider when designing and developing CT-based systems for K-12 education, and for analyzing existing CT-based environments for pedagogical applications.

CT is at the heart of the CoLPACT framework and we use Brennan & Resnick's (2012) three-dimensional framework comprising computational concepts, practices, and perspectives (see Section 2.1) to capture and analyze the different aspects of CT fostered in different learning environments. Along with identifying the target CT skills, deciding the context or the domain in which the CT skills will be taught is fundamental to designing a CT-based system. Generating contexts that are personally relevant and meaningful to learners and/or provide bridges to real-world applications help motivate learners and broaden their participation in such systems. At the K-12 level, robotics (Martin et. al., 2013), game design (Kafai, 1995, Repenning et.al., 2010), storytelling (Kelleher, 2008), designing mobile apps (Wolber et. al., 2011), programming multimedia applications (Forte & Guzdial, 2004), and integration of CT with different school subjects, such as science (Basu et. al., 2012; Repenning et.al., 2010; Allan et.al., 2010; Jona et. al., 2014), language arts (Buechley et. al., 2013), as well as music and art (Disalvo & Bruckman, 2011) are examples of contexts that have been employed in broadening students' participation in and perceptions of programming. The context employed is likely to be guided by the target audience and the goals of the learning environment, and may in turn influence the choice of language and pedagogy used in fostering CT skills.

The computational language used is an important aspect of a CT-based learning environment and is likely to be influenced by the context in which CT skills are fostered in the environment including the target audience for the environment. The language is used to point out CT concepts to students and students use it to apply CT concepts and practices to generate computational artifacts like games, digital stories, models and simulations. For example, if the context is a curricular discipline, like science or mathematics, a domain-specific programming language (Sengupta et. al., 2013) that emphasizes the domain concepts along with highlighting the generality of CT concepts across domains could be more appropriate as opposed to a general-purpose programming language. Also, systems could employ visual programming languages versus text-based or graphical ones, or agent-based modeling languages versus object-oriented or system-based ones based on the age group of the target audience or the goal of the learning environment. Visual drag-and-drop languages are believed to alleviate syntax problems that young students face when assembling programs or computational models (Soloway, Guzdial, & Hay, 1994). Some research has also claimed that visual programming languages can make the understanding of computational processes more accessible by making the logic associated with complex elements of flow of control, such as loops and conditionals, easier to grasp (Parsons & Haden, 2007). Similarly, agent-based modeling languages are considered more intuitive for younger children and serve as powerful representations when CT is promoted in the context of complex systems. We use the agent-based modeling paradigm in building our CT-based learning environment and review important research in this area in Section 3.2.

Like the computational language used, the pedagogy guiding the design of a CT-based learning environment is also likely to be influenced by the target users and expected setting for the environment. Several CT-based environments follow the learning-by-design pedagogy, where students design and build their own computational models for curriculum-related topics, games, and stories. Examples of other pedagogies followed include learning by remixing and reusing code, learning by debugging, and collaborative learning. For example, simple remixing of given code might require just a few mouse-clicks to copy programs and not involve thinking computationally, but selective remixing and deciding what to modify in selected code segments versus what to keep and where to add or delete procedures or variables within a program require a deep understanding of CT skills (Kafai & Burke, 2010). Learning by debugging involves a thorough understanding of computational processes – how the control flows and how variables are updated – in order to detect bugs in the given code and correct them generating a computational artifact with the desired behavior. Collaborative learning or computational participation (Kafai &Burke, 2010) generally associates CT-based environments with open source sites in the style of communities where learners can share, comment on, and contribute to each other's coded creations. This learning paradigm builds on insights from educational research that fruitful learning is not done in isolation but in conjunction with others. Also, learning how to program collaboratively involves other CT elements, such as decomposing complex tasks between collaborators and coordinating control flow between different components.

Assessments for measuring students' CT skills, the final dimension of the CoLPACT framework, are believed to be quintessential for a CT-based environment to be integrated into the K-12 curriculum (NRC, 2011), but standardized CT assessment are still lacking (Grover & Pea, 2013; Basu, Kinnebrew, & Biswas, 2014). Assessments in several environments focus on studying the frequency of different computational concepts students use in their computational models and how the frequencies vary with time. More frequent use of CT concepts like loops and conditionals is favored, irrespective of the correctness of the final computational artifacts designed. Assessment of artifacts and computational practices employed during construction of the artifacts is generally performed using researchers' observation notes (Kafai et. al., 2013) and artifact-based interviews (Brennan & Resnick, 2012), which are logged and analyzed. Some environments also include pre- and post-assessments for testing algorithmic thinking and use of abstractions by making students perform small modeling and debugging tasks. Some of these assessments are dependent on the specifics of the system and are, hence not administered before the intervention, making them non-generalizable, and, thus making pre-post comparisons impossible. Very few environments include generalizable pre-post tests for assessing computational concepts and problem solving skills, especially in the context of STEM problems (Basu et. al., 2014; Jona et. al., 2014). Some environments also only include assessments of computational perspectives, which are generally conducted through pre-post surveys focusing on measuring changes in students' attitude towards and awareness about the term CT, without measuring proficiency in CT skills.

We believe that the CoLPACT framework acts a suitable platform for characterizing and comparing different CT-based learning environments and provides valuable pointers for the design of new CT learning environments. The five dimensions of the framework are not completely independent; choices made with respect to one dimension can influence choices made in other dimensions. For example, the choice of modeling language and pedagogy can vary between those used with younger children and the ones employed with high school students, or based on whether the learning context is open-ended or constrained. Similarly, curricular contexts tend to have a stronger emphasis on assessments, though it has been seen that when the curricular context is something other than CS, assessments tend to focus more on the curricular topic, rather than CT concepts and practices. In the next section, we review some well-known CT-based learning environments and their use in K-12 using the CoLPACT framework.

2.3.2 Using the CoLPACT framework to review CT-based learning environments

In this section, we use the CoLPACT framework to analyze some of the widely used CT-systems. We limit our reviews to CT-based learning environments, which have been and are used primarily in K-12 settings. We study both curricular and extra-curricular use of these environments, since a majority of the well-known CT-based environments are still used primarily as part of extra-curricular activities.

Scratch (Resnick, 2007) is one of the most popular and widely used CT-based systems today. Scratch users are primarily K-12 students between the ages of 8 and 16, though Scratch has been used in some introductory-level college courses as well. Scratch uses a general-purpose, visual block-based language where students snap together different blocks to generate their own computational models or programs. Since Scratch is inspired by how students build with Lego blocks, its blocks have connectors like Lego bricks, suggesting how the blocks fit together to describe a larger system. The blocks are shaped to fit together only in ways that make syntactic sense. Scratch also allows users to personalize their programs by importing photos and music clips, recording voices, and creating graphics (Resnick et. al., 2009). Though Scratch itself uses a general-purpose language, extensions are being currently developed that will enable anyone to extend the vocabulary of the Scratch programming language through custom programming blocks including domain-specific programming primitives written in JavaScript (Dasgupta et. al., 2015).

Scratch is an open-ended system and not tied to any learning domain (context) in particular. Students' working with the system have generated diverse products that include video games, stories, interactive newsletters, science simulations, virtual tours, birthday cards, and animated dance contests. Currently, Scratch has primarily been used in after-school computer clubs and summer camps as an open-ended CT-based environment not tied to any specific curricular topic in order to introduce students to programming and foster their creative thinking skills. When Scratch is used in-class, its primary use has been in elective computer classes. Various context-independent curricula have been developed revolving around the use of Scratch by different researchers. Leading the effort is the ScratchEd team, whose Scratch curriculum guide provides an introduction to creative computing with Scratch, using a design-based learning approach (http://scratched.gse.harvard.edu/resources/scratch-curriculum-guide). The guide is organized as a series of twenty 60-minute sessions, and includes session plans, handouts, projects, and videos. Each session is organized into 5 topics: introduction, arts, stories, games, and final project. The guide is both subject-neutral and grade-neutral to accommodate different settings for any teacher who wants to support students' development of CT through explorations with Scratch.

Irrespective of the context, programs involve use of some subset of the following seven computational concepts – *sequences, loops, parallelism, events, conditionals, operators,* and *data*. Scratch claims to foster four main sets of computational practices: *being incremental and iterative*, *testing and debugging*, *reusing and remixing*, and *abstracting and modularizing*; and three primary computational perspectives: *being expressive, questioning, and connecting with others* (Brennan & Resnick, 2012). Along with the learning-by-design pedagogy, Scratch is also largely based on the collaborative learning pedagogy allowing students to reuse and remix others' programs. It has developed a vibrant online community, where Scratchers worldwide share their programs and comment on and remix existing programs.

Scratch itself does not include assessment for CT skills. Assessments with Scratch tend to vary based on the setting in which it is used. When used in curricular contexts, assessments tend to include pre- and post-tests on computational definitions and students' understanding of basic CT concepts like loops, sequences and conditionals (Grover, Cooper & Pea, 2014). Other forms of assessments involve interviewing students about their programs and computational perspectives, studying use of CT concepts in students' Scratch programs over time (Brennan &

Resnick, 2012), and measuring students' abilities to transfer CT skills from Scratch to text-based languages (Grover, Pea & Cooper, 2014).

Alice is another well-known CT-based learning environment that uses the context of creating animations and storytelling using 3D models to promote CT skills (Pausch et. al., 1995). Alice also uses a visual drag-and-drop language, but follows an object-based, event-driven programming paradigm. Learners place objects from Alice's gallery into the virtual world that they have imagined, and then program each of the objects' properties and event-based behaviors. Alice also targets the same computational concepts, practices and perspectives as Scratch. Though the initial versions of Alice (Alice 1.0 and Alice 2.0) were developed for and used by high school and first-year college students, **Storytelling Alice (SA)** was later developed and tested with middle school students to tap into storytelling's motivational effects for younger students (Kelleher & Pausch, 2007). SA contains a gallery of 3D characters and scenery with custom animations, allowing users to program social interactions between characters using high-level animations. Using a context like storytelling has been reported to be particularly beneficial in increasing interest in programming among girls and women. Alice 3.0 is again geared towards elder students, allowing them to switch between Java programming and object-oriented drag-and-drop programming.

In general, Alice adopts a learning-by-design pedagogy with some studies following a use-modify-create cycle (Werner, Denner & Campe, 2012) where learners first understand provided Alice programs and modify them as needed before creating their own programs. Alice also has its own online community, like Scratch where learners can share and collaborate on their Alice animations and stories. Also, similar to Scratch, Alice is used predominantly in after-school settings with K-12 students, and less often in curricular settings. But, instructional materials, tutorials videos, and textbooks have been developed as part of a CS curriculum using Alice, and are readily available for educators who choose to use Alice to teach CT concepts.

The *Fairy Assessment* (Werner, Denner & Campe, 2012) has been developed for Alice and requires students to modify or add methods to existing code to accomplish given tasks, thus displaying their understanding of abstraction, conditional logic, algorithmic thinking, and other CT concepts to solve problems. However, this assessment is Alice-based and requires subjective and time-consuming grading (Grover, Cooper & Pea, 2014). Other multiple-choice assessments have been used for measuring learning of Alice programming concepts (Moskal, Lurie, & Cooper, 2004), but not in K-12 settings. Recently, Cooper, Nam, & Si (2012) incorporated an Intelligent Tutoring System (ITS) into the Alice environment using stencils (Kelleher & Pausch, 2005), a graphical overlay system, to detect students' mastery of different CT skills and interactively provide instruction. The ITS tries to help students master a set of CT skills by providing tutorials for each, where tutorials are built as a series of missteps, forcing students to make common errors associated with the use of the skills, and observe the outcomes. For each CT skill, a tutorial is followed by an alternate activity assessing the same skill. The alternate assessment activities involve showing students a YouTube video that represents the correct desired solution and asks students to write code corresponding to the desired solution. Students' solutions are assessed using stencils, which Cooper, Nam, & Si (2012) point out is somewhat problematic, since it requires students' code to be an exact match to the solution specified in the stencil, and results in correct student code sometimes being flagged as incorrect.

AgentSheets (Repenning, 2000) is another CT-based system that teaches students programming and CT skills through game design and creation of simulations. It also adopts a visual drag-and-drop language, but the language is based on an agent-based modeling paradigm (Wilensky & Resnick, 1999; Chi, 2005). Similar to a spreadsheet, an agentsheet is a computational grid, but the grid comprises agents in this case. Users create the different type of agents and specify how each agent looks and behaves in different situations. All agent behaviors are implemented using "If-Then" conditional statements (Koh et. al., 2010). AgentSheets enables the use of 16 different conditions and 23 different actions, in combination, to create behaviors for any given agent. It allows up to tens of thousands of agents, thus making it suitable for modeling and exploring complex scientific phenomena like ecological systems, spread of diseases, etc. This ability to support games as well as computational science applications distinguishes AgentSheets from Scratch and Alice, and makes it suitable for use in both CS and STEM education.

AgentSheets supports a middle and high school curriculum, called Scalable Game Design (Repenning, Webb & Ioannidaou, 2010) aligned with the ISTE National Educational Technology Standards (NETS). This project aims to motivate all students including women and underrepresented communities to learn about computer science through game design starting at the middle school level. The project consists of two modules. In 6th grade a one-week module is integrated into an existing required course. In 7th grade a four-week module in elective courses allows students to move on to more complex games or computational science simulations.

Assessments using Agentsheets have not been sufficiently documented in the literature. Metrics, like Program Behavior Similarity (PBS) and tools, like the Computational Thinking Pattern (CTP) graph have been developed to provide students with instant semantic evaluation feedback after submitting their models. These metrics help detect transfer of CT skills between Agentsheets activities and across activities in related disciplines (Koh et. al., 2010). In addition, Retention of Flow is used as an evaluation measure of students' engagement in open ended Scalable Game Design activities or similar 'Hour of Code' game-design activities (Repenning et. al., 2016). Student retention in an activity is calculated based on the program length or lines of code a student has created in the activity, where lines of code correspond to methods, rules, conditions, and actions specified for agents described in the activity.

Agentsheets also differs from most existing systems in that it attempts to scaffold students as they work in the environment, though the scaffolding is somewhat elementary. For example, students receive feedback about the computational models or games they build through representations like the CTP graph (Koh et. al., 2010), which help them realize CT patterns missing in their models. Besides students, AgentSheets also provides real-time assessment of students' models to scaffold teachers and provide them with information about individual student progress so they can help student who are struggling with their AgentSheets activities. A cyberlearning system titled REACT (Real-time Evaluation and Assessment of Computational Thinking) provides teachers with a sortable dashboard, consisting of data from each student, that shows the characters students created and used to populate their game or simulation world as well as the semantic meaning behind what students have programmed (Basawapatna, Repenning, & Koh, 2015). Further, AgentSheets 3 provides conversational programming to help students understand the meaning and semantics of the program they are working on.

Other attempts to integrate CT and STEM include EcoScienceWorks (Allan, et.al., 2010), the Computational Thinking in STEM project (Jona et. al., 2014), and Project GUTS (http://projectguts.org/). EcoScienceWorks (ESW) is an ecology curriculum built for 7th grade students across the state of Maine that includes targeted simulations for concepts like succession, habitat fragmentation, species interactions, and invasive species, along with a code block programming module called *Program a Bunny*. Since the goal of the project is to increase student interest in CT and computer programming within the constraints of the middle school curriculum (where computer programming is not taught), the project re-designed existing computer simulations in ecology and added the capability for students to program their own simulations. Student assessments in ESW measure students' CT perspectives and comprise of observations of student behavior, student interviews, and online surveys. The Computational Thinking in STEM (CT-STEM) project, on the other hand, emphasizes learning CT concepts and practices along with increasing students' interest in CT. such that they later choose CT and CS courses, and perhaps a future CT career trajectory. CT activities are embedded in the context of existing high school STEM courses in Physics (projectile motion, Ohm's law, resonance), Mathematics (probability, exponential functions, fitting real data), Chemistry (radioactivity, rusting, gas laws), and Biology (predator-prey relations, DNA sequencing, genetics, cell structures). CT-STEM relies heavily on NetLogo – an agent based modeling and simulation platform with a textual language (Wilensky, 1999). Students work with existing PhET (Wieman et. al., 2008) and NetLogo models, and also create their own computational models from scratch. The project emphasizes learning data analysis, computational problem solving, and systems thinking in the context of modeling and simulations, and includes assessments encompassing each of these CT-STEM skills. Project GUTS (Growing Up Thinking Scientifically) is another example of an attempt to integrate CT and STEM. It is a summer and after-school STEM program for middle school students. Recently, Project GUTS has partnered with Code.org to develop a middle school science program that consists of curricular units to introduce CT in the context of life, physical and earth sciences, while addressing course standards. Students go through a use-modify-create cycle to learn about curricular science topics like the global climate system, ecosystems and chemical reactions, which are modeled as complex systems using StarLogo – an agent-based modeling language. The goal of the program is to prepare students to pursue formal CS courses during high school. Hence, assessments focus on student interest in CT, besides curricular assessments for science.

Efforts have also been made to introduce K-12 students to CT skills in more authentic and tangible contexts. For example, IPRO (I can PROgram) is an iOS application designed to teach high school students how to program in a mobile, social programming environment (Martin et. al., 2013). Students program virtual robots to play soccer on their iPod Touch and are also provided a full size physical replica of the agent environment in the classroom. They work in teams to program, test and debug their virtual robots, and engage in matches between their team's robots and other teams'. IPRO uses a Scheme-based programming language consisting of a library of sensors and actions, connected by conditional statements. Conditional statements can be endlessly nested and all possible programs are executable (i.e., no syntax errors are possible). Studies with high school students using IPRO have used attitudinal survey questions, learning questions and transfer questions as assessments. The learning questions are specific to programming in IPRO, while the transfer questions test predicate calculus skills which are important for understanding programming. Results have shown improved performance on programming content and computational content transfer, and improved attitudes towards programming and CS. Electronic textiles or e-textiles (Kafai et. al., 2013) is another project that uses tangible construction kits to teach programming and engineering concepts to high school students. E-textiles involves using the Lilypad Arduino language in conjunction with sewable microcontrollers, sensors and actuators. A primary goal of the project is to broaden participation and perceptions about computing using a context that can be especially appealing to women. The e-textiles curriculum focuses on both the basics of circuit designs and learning how to program using Lilypad Arduino. In a study with high school students, students' completed artifacts, observations of their design approaches, and student interviews were analyzed to assess CT concepts, practices, and shifts in perspectives. The Arduino program codes for all projects were found to contain key CT concepts such as sequences, loops, conditionals, operators, and variables. Students mostly engaged in iterative CT practices of imagining and designing and constructing a little bit, then trying it out, and then developing it further, and they started considering it as a more personally relevant task.

Several other applications have been and continue to be developed for promoting CT skills and interest in coding and computer science, especially for the female population of students. Many of them rely on a visual editor called Blockly to create a great UI for novice users (https://developers.google.com/blockly/?hl=en). Blockly allows users to write programs by plugging blocks together, and developers can integrate the Blockly editor easily into their own web applications. For example, the MIT App Inventor (http://appinventor.mit.edu/explore/) uses Blockly to provide an innovative beginner's introduction to programming and app creation that transforms the complex language of text-based coding into visual, drag-and-drop building blocks. The simple graphical interface claims to grant even an inexperienced novice the ability to create a basic, fully functional app within an hour or less, thus empowering all people, especially young people, to transition from being consumers of technology to becoming creators of it. Made with Code (https://www.madewithcode.com/) is another application that uses Blockly to encourage girls to code in diverse fields ranging from music to fashion. Some other applications that rely on

Blockly to create their visual interfaces for promoting different CT skills are Blockly Games - Games for tomorrow's programmers, Gamefroot - Make, play and share games; OzoBlockly - Programming line-following robots, and Wonder Workshop - Robots for play and education.

2.4 Critical Summary of existing CT based learning environments

Table 1 summarizes the state of the art of CT-based environments by characterizing some of the common environments along the five dimensions of the CoLPACT framework.

Table 1. A review of existing CT-based environments using the CoLPACT framework

CT-based environments	**Context (Co)**	**Modeling Language (L)**	**Pedagogy (P)**	**Assessment (A)**	**Computational Thinking (CT)**
Scratch	Open-ended; After-school computer clubs; Summer camps; Elective computer classes in K-12; College-level CS courses	General-purpose; Visual, block-based	Learning-by-design; Collaborative learning; Learning through reuse and remix	Pre- and post-tests on definitions and use of CT concepts; Interviews about CT perspectives; Use of CT concepts in Scratch programs over time; Transfer of CT skills from Scratch to text-based languages	Concepts like sequences, loops, parallelism, events, conditionals, operators, and data; Practices of being incremental and iterative, testing and debugging, reusing and remixing, and abstracting and modularizing; Perspectives of being expressive, questioning, and connecting with others
Alice	Creating animations and storytelling using 3D models; After-school computer clubs; Summer camps; Elective computer classes in K-12; College-level CS courses	General-purpose; Visual drag-and-drop language; Object-based; Event-driven	Learning-by-design based on a use-modify-create cycle; Collaborative learning; Learning through reuse and remix	System-specific *Fairy Assessment* requiring students to modify or add methods to existing code to accomplish given tasks	Concepts of sequences, loops, parallelism, events, conditionals, operators, and data; Practices of being incremental and iterative, testing and debugging, reusing and remixing, and abstracting and modularizing; Perspectives of being expressive, questioning, and connecting with others

AgentSheets	Game design and creation of simulations; Middle and high school curriculum called Scalable Game Design	General purpose; Visual drag-and-drop language; Agent-based modeling paradigm	Learning-by-design; Conversational programming; Instant semantic evaluation feedback through CTP graphs	Program Behavior Similarity and the CTP graph to detect transfer of CT skills	Concepts of sequences, loops, parallelism, conditionals, operators, and data; Practices of being incremental and iterative, testing and debugging, and abstracting and modularizing; Perspectives of being expressive, questioning
ESW	Ecology curriculum for 7th grade students across the state of Maine	General purpose; Agent-based modeling paradigm	Learning by re-designing existing computer simulations	Measure CT perspectives through observations of student behavior, student interviews and online surveys	Concepts – sequences, loops, conditionals, operators; Practices - being incremental and iterative, testing and debugging, and abstracting; Perspectives - being expressive, questioning
CT-STEM	High school STEM courses in Physics, Mathematics, Chemistry, and Biology	General purpose; Agent-based modeling paradigm; Text-based language	Learning using simulations; Learning-by-design	Assessments for CT concepts, computational modeling and problem solving, data analysis	Data analysis, computational problem solving, and systems thinking in the context of modeling and simulations
Project GUTS	Summer and after-school STEM program for middle school students; Middle school curricular units in life, physical and earth sciences	General-purpose; Block-based; Agent-based modeling language;	Learning-by-design through use-modify-create cycles	Assessments for student interest in CT; curricular assessments for science	CT concepts of loops, conditionals, sequences, variables; CT perspectives of readiness to pursue formal CS courses during high school
IPRO	Playing soccer with virtual robots using iPod Touch in elective high school classes	Scheme-based programming language consisting of a library of sensors and actions	Learning-by-design; Collaborative learning; Engage in inter-robot matches	Attitudinal survey questions; IPRO-specific learning questions; transfer questions testing predicate calculus skills	Concepts like conditionals, loops, and variables; Sense-act cycles; Computational modeling and problem solving

e-textiles	Circuit-design and sewing in high school classrooms	Lilypad Arduino language in conjunction with sewable microcontrollers, sensors and actuators	Learning-by-design; Engage in personally relevant context	Study completed artifacts; Observe design approaches; Interview students	Concepts like sequences, loops, conditionals, operators, and variables; Practices of iterative imagining, designing, constructing, testing and refining; Perspectives of considering computing more personally relevant

We see that in spite of recognizing the need for integrating CT with the K-12 curricula, and the several suggestions discussed in various workshops and committees about how to achieve this, several CT-based learning environments today still involve extracurricular participation through summer camps, after-school computer clubs, and elective curricular participation. Several of these environments do not try to connect their activities and learning goals to existing K-12 standards or STEM learning concepts. Broadening participation in CS through motivational extracurricular CT-based activities may be a good first step, but CT eventually needs to be integrated into the K-12 curricula, possibly introducing it in middle school, since it is the age at which students start deciding on future career choices based on their assessments of their skills and aptitudes. Also, curricular integration will help remove the variables of self-selection, confidence, and willingness to opt for elective and extra-curricular programs from the equation. As a result, all students, including minorities and women will be necessarily exposed to CS-related concepts. The successful and sustained integration of CT concepts and skills into the K-12 curriculum requires providing K-12 teachers with learning environments and other resources demonstrating how to integrate CT with existing grade-relevant curricular topics, while keeping the learning overhead low (both for teachers and students).

We notice that CT-based environments that are used in K-12 curricular contexts generally integrate CT with existing science topics, since it is not always feasible to accommodate CS curricula independently into existing K-12 curricula. In environments that integrate CT with existing K-12 science curricula, the choice of computational language and pedagogy employed appear to be similar. We find that environments promoting CT in the context of K-12 science topics (for example, AgentSheets, ESW, CT-STEM, and Project GUTS) often employ an agent-based modeling paradigm, since it is believed to aid and scaffold students' understanding of complex science topics. Also, such environments mostly employ the learning-by-design pedagogy, sometimes as part of a use-modify-create cycle, and generally make students work independently, perhaps because collaboration and reusing of code make grading of curricular assessments difficult. As part of this dissertation research, we will develop our learning environment to simultaneously foster CT skills and science learning at the K-12 level. We also adopt an agent-based modeling paradigm and an independent learning-by-design pedagogy involving build-test-refine cycles in the context of model building of science processes with support for model verification. The detailed design and implementation of our learning environment is provided in Chapter 3.

We see in Table 1 that even for environments that attempt to integrate CT with existing K-12 science curricula, assessments for measuring CT concepts and skills are severely lacking. Assessments in such environments generally focus on students' interests in computing and their likelihood of pursuing CT courses in the future, while existing curricular assessments measure science learning. Currently, among the environments that integrate CT with science learning, CT-STEM is the only one to have developed pre-post assessments for measuring students' computational thinking and problem solving skills. However, even CT-STEM does not assess the computational models built by students for correctness or use of computational constructs. Assessing students' knowledge of CT concepts and their computational models is somewhat more common when the context is open-ended or related to elective CS courses. Even in such cases, models are generally not evaluated for correctness, but merely for frequency of use of different computational constructs. Thus, we realize that attention needs to be paid in CT-based learning environments towards developing standardized, objective, and holistic assessments for CT, which includes developing metrics for assessing students' computational models, and their learning behaviors in terms of use of CT practices. In addition to the lack of well-developed CT assessments in existing CT-based learning environments, the developmental processes of students as they are introduced to CT is little understood, and the challenges they face is not known. Attention needs to be paid to develop methods that detect and scaffold these challenges.

In this dissertation research, we address the aforementioned challenges by designing and developing Computational Thinking using Simulation and Modeling (CTSIM) - a CT-based science learning environment, building a learning progression using CTSiM for middle school students, and developing holistic assessments for measuring students' CT proficiency, science learning and modeling skills. Using interview-based observational assessments with students using

CTSiM, we identify and categorize students' challenges when they work in such an environment, and then design a scaffolding strategy to help students deal with some of their challenges. Finally, we demonstrate the effectiveness of our scaffolding approach through a controlled research study using different assessment metrics we have developed based on students' actions in the CTSiM learning environment and offline assessments administered outside CTSiM.

CHAPTER 3

Initial design and development of the CTSiM learning environment

In this chapter, we discuss the initial design, and implementation of the Computational Thinking using Simulation and Modeling (CTSiM) learning environment, along with the sequence of learning activities we have designed for synergistic learning of CT and middle school science. The design of CTSiM is informed by lessons learned from previous research in learning-by-design, educational computing, and multi-agent based modeling (Basu et. al., 2012; Sengupta et. al., 2013; Basu et. al., 2013). We discuss previous research and key design principles guiding the integration of CT and science learning in CTSiM in Sections 3.1-3.4, and describe the initial CTSiM architecture and interfaces, the selected curricular science topics, and the learning activity progression involving these topics in Sections 3.5-3.7.

3.1 Design as a core focus of science learning using computational modeling

In Section 2.1, we discussed the importance of '*computational doing*', and how CT is considered to become evident in the form of design-based epistemic and representational practices (Hemmendinger, 2010; Sengupta et. al., 2013). Grover and Pea (2013) have identified examples of representational practices as abstractions and pattern generalizations (that include modeling and simulation activities); symbol systems and representations; algorithmic notions of flow of control; structured problem decomposition (modularizing); conditional logic; and iterative, recursive, and parallel thinking. Other epistemic practices include systematic processing of information; adopting efficiency and performance constraints; and debugging and systematic error detection. This '*computational doing*' perspective, in turn, aligns with the *learning-by-design* pedagogy in general, and with the *science-as-practice* perspective, in particular.

The *learning-by-design* pedagogy applies to all domains and suggests that students learn best when they engage in the design and consequential use of external representations for modeling and reasoning (Blikstein & Wilensky, 2009; Kolodner *et al.,* 2003; Papert, 1991). This pedagogy promotes active learning and greater agency for the learner by activating eight knowledge processes that represent distinct ways of generating knowledge and learning: experiencing the known and the new, conceptualizing by naming and by theorizing, analyzing functionally and critically, and applying creatively and appropriately (Healy, 2008). The *science as practice* perspective (Duschl, Schweingruber, & Shouse, 2007; Lehrer & Schauble, 2006), on the other hand, emphasizes the importance of engaging learners in the development of epistemic and representational practices to help them understand how scientific knowledge is constructed, evaluated, and communicated. Modeling – the collective action of developing, testing, and refining models - has been described as the core epistemic and representational practice in the sciences (Lehrer & Schauble, 2006; NRC, 2010). Modeling involves carefully selecting aspects of the phenomenon to be modeled, identifying relevant variables, developing formal representations, and verifying and validating the representations. Hence, developing a model of a scientific phenomenon involves key aspects of CT: identifying appropriate abstractions, and iteratively refining the model through debugging and validation against expert or real world data.

CTSiM adopts this learning-by-design pedagogical approach which interweaves action and reflection by engaging students in cycles of building computational models of given science phenomena, observing the behavior of their models as simulations, validating their simulations against provided correct simulations, and refining their models accordingly.

3.2 Agent based modeling and simulation for learning science

Agent-based modeling (ABM) is a form of computational modeling in which individual entities in a complex system (the agents) are modeled as computational objects with specific rules defining their behavior and their interactions with other agents. For example, fish can be considered an agent-type in the simulation of a pond ecosystem, while electrons can be agents when modeling and simulating the flow of electricity. Models can comprise a single agent-type or multiple types of agents, in which case they are known as Multi-Agent Based Models or MABMs (Macal & North, 2008). The collective interactions of the agents, each concurrently acting out its behavioral rules, may be used to generate known phenomena, or reveal new, emergent behaviors of the complex system (Wilensky, Brady & Horn, 2014). In other words, MABMs provide a framework for (1) understanding and explaining how system-level behaviors emerge from individual agent behaviors, and (2) what-if analyses, i.e., how perturbations in the system can affect and alter overall system behaviors. In Sections 3.2.1-3.2.3, we describe the use of ABMs and agent-based simulations as powerful tools to introduce CT across the K-12 science curriculum.

3.2.1 Pedagogical significance of agent based modeling

A wide variety of scientific phenomena can be studied and analyzed using a complex systems framework (Holland 1995; Kauffman 1995; Goldstone & Wilensky, 2009), where the collective, global behavior of the system emerges from the properties of individual elements and their interactions with each other. The global or macro behaviors - known as *emergent* phenomena - are often, not easily explained by the properties of the individual elements (e.g., Bar-Yam 1997, p. 10; Holland 1998). Emergent phenomena are central to several domains, such as population dynamics, natural selection and evolution in biology, behavior of markets in economics, chemical reactions, and statistical mechanics, thermodynamics and electromagnetism in physics (Mitchell 2009; Darwin 1871; Smith 1977; Maxwell 1871). For example, in chemistry and physics, gas molecules' elastic collisions at the micro level, produce the macro-level properties of pressure and temperature. In biology, animals interact with others of the same and different species, and the environment to survive, grow, and reproduce at the individual level that leads to phenomena, such as evolution, natural selection, and population dynamics at the ecosystem level (Wilensky, Brady & Horn, 2014).

A number of studies have shown that students experience difficulty in understanding emergent phenomena in science (Hmelo-Silver & Pfeffer, 2004; Jacobson, 2001; Wilensky & Resnick, 1999; Chi, 2005). Agent based modeling holds immense potential to support learning of complex science phenomena, since it provides the means to build on students' intuitive understandings about individual agents acting at the micro level to grasp the mechanisms of emergence at the aggregate, macro level. When students interact with, or construct a MABM, they initially engage in intuitive "*agent-level thinking*" (i.e., thinking about the actions and behavior of individual actors in the system) (Goldstone & Wilensky 2008). Thereafter, with proper scaffolding, students can build upon their agent level understanding and develop an understanding of aggregate-level or emergent outcomes by interacting with multiple complementary representations of the putative phenomena: agent-level rules and variables, dynamic visualization that simultaneously displays actions of all the agents in the microworld, and graphs that show aggregate level patterns over time (Tan & Biswas 2007; Wilensky & Reisman 2006; Blikstein & Wilensky, 2009; Klopfer, 2003; Danish *et al.*, 2011; Dickes & Sengupta 2013).

In several science classrooms, differential equations and other mathematical formulae still form the most commonly used aggregate-level formalisms for teaching students how different aggregate variables evolve over time (Wilensky & Reisman 2006). While mathematically sophisticated, these formalisms do not make explicit the underlying agent-level attributes and interactions in the system, and therefore, remain out of reach of most elementary, middle, and even high school students. It has been suggested that the lack of connection between students' natural, embodied, agent-based reasoning, and the aggregate forms of reasoning and representations they encounter in school creates a barrier to their understanding of emergent phenomena. When children construct or use MABMs to learn complex scientific phenomena, this divide can be bridged (Dillenbourg, 1999; Jacobson & Wilensky, 2006; Tisue & Wilensky, 2004; Sengupta & Wilensky, 2009; Goldstone & Wilensky, 2008; Dickes & Sengupta, 2013; Basu, Sengupta & Biswas, 2014). Constructing and running their models helps students realize the implications of their ideas, provokes new conjectures, and drives motivating cycles of debugging involving modeling, execution, and refinement (Wilensky, Brady & Horn, 2014). Also, students can use their models to explore what-if questions by varying initial conditions and model parameters, or modifying existing behavioral rules of the agents in the model. Thus, working in MABM environments helps prepare them for authentic inquiry in the scientific disciplines.

3.2.2 A review of studies and environments using ABM for learning science

One of the earliest and best-known agent-based programming languages is LOGO (Papert, 1980). LOGO users learn fractions and other important STEM concepts by programming the behavior of a protean agent—the LOGO turtle. The Logo turtle is considered to be body syntonic, meaning that understanding the behaviors and the rules guiding the behaviors of a turtle is related to learners' understandings of their own bodies. This body-syntonicity is believed to help young leaners to bootstrap their intuitive knowledge in order to learn canonical science concepts (Sengupta & Farris, 2012).

The LOGO language has been used and extended over the years in different MABM environments, like StarLogo (Resnick, 1996) and NetLogo (Wilensky, 1999). Unlike LOGO, which allows only a few turtles, these environments allow for modeling of thousands of agents or turtles, that can perform their actions concurrently. Having such large number of turtles facilitates the modeling of several types of complex systems, where the system behavior may show qualitative changes as the turtle population is changed. Also, turtles in these environments have better sensing properties than the traditional LOGO turtles, thus they are capable of executing more complex behaviors based on interactions with other turtles and the environment. Further, the turtles' environments are also modeled as agents (known as *patches*) in these agent-based environments. Patches are static, but have many of the same capabilities as turtles. For example, each patch could diffuse some of its "chemical" into neighboring patches, or it could grow "food" based on the level of chemical within its borders (Resnick, 1996).

The current version of StarLogo - StarLogo TNG (The Next Generation) – attempts to lower the barrier to entry for programming by adding a block-based programming interface to the agent-based modeling and simulation environment (Klopfer et. al., 2009). StarLogo TNG also includes 3D graphics and sound to attract more young people into programming and creation of richer games and simulation models. Unlike StarLogo TNG, NetLogo (Wilensky, 1999) employs a written, text-based programming language. This is why StarLogo TNG is sometimes considered more user-friendly for younger users, who can focus on agent-based programming without being bogged down by spelling and syntax errors. However, NetLogo also has its set of added advantages over StarLogo TNG. NetLogo can handle more agents, model more complex behaviors, and simulation runs are more time-efficient. The NetLogo language is more extensive with better support for lists, agent sets, and local variables, and it is extendable and controllable via Java, making it better suited for embedding in Java-based learning environments (http://projectguts.org/). In CTSiM, we integrate the benefits of each of these agent-based environments by developing our own block-based programming interface for constructing agent-based computational models and using NetLogo under the hood to generate simulations corresponding to the models.

The described MABM environments are used extensively in elementary through undergraduate classrooms to help students learn about emergent phenomena in different science disciplines. For example, NetLogo comes with a large model library, which can form the basis for curricular material in several science topics. Many middle school students use the NetLogo *Fire* model to study natural disasters, such as volcanoes and forest fires, while several high school students use models from the NetLogo GasLab suite to understand Kinetic Molecular Theory and effects of molecular interactions on aggregate properties of gases like pressure and temperature. NIELS (NetLogo Investigations in Electromagnetism), a curriculum of multi-agent computational models, has been shown to help undergraduate students develop a deep, expert-like understanding of phenomena, such as electric current and resistance by modeling them as phenomena that emerge from simple interactions between electrons and other charges in a circuit (Sengupta & Wilensky, 2009). ViMAP (Sengupta & Farris, 2012) is an example of a multi-agent environment based on the NetLogo modeling platform. In ViMAP, students can construct a new NetLogo simulation model or modify an existing NetLogo simulation model using visual and tactile programming. Students can also dynamically generate inscriptions (e.g., graphs) of aggregate level phenomena using the MAGG (Measuring Aggregation) functionality. Studies using ViMAP to teach Kinematics have shown that elementary school students develop a deep conceptual understanding of the relevant physics concepts involved.

On the other hand, Project GUTS (Growing Up Thinking Scientifically) is an example of a year-long STEM program for middle school students that uses StarLogo to engage students in agent-based modeling and scientific inquiry. During the first four weeks of the program, students learn about complex systems and how to create computer models from scratch. Subsequently, during each six-week afterschool unit, students investigate a problem, interview experts and community members, gather data, and run experiments on their computer models to better understand the problem being studied (http://projectguts.org/). Then, students upload their investigations to the project website and compare and discuss their models with others. Recently, Code.org and Project GUTS have partnered to deliver a middle school science program consisting of four instructional modules and professional development for the introduction of CS concepts into science classrooms within the context of modeling and simulation. Details about this collaborative project have already been presented in Section 2.3.2.

Though MABMs have been shown to be effective pedagogical tools in learning about emergent phenomena in various domains (Chi, 2005; Mataric 1993; Wilensky & Reisman, 2006), students generally require scaffolding when learning using MABMs. For example, Wilensky and Reisman (2006) showed that high school students were able to develop a deep understanding of population dynamics in a predator-prey ecosystem by building MABMs of wolf-sheep predation, but they needed to be provided explicit assistance in terms of programming support and reflection prompts by the interviewer. Tan and Biswas (2007) reported a study where 6th grade students were scaffolded by the interviewer while using a multi-agent based simulation to conduct science experiments related to a fish tank ecosystem. The experimental study showed that students who used the simulation showed significantly higher pre-post test learning gains, as compared to a control group that were provided with the results of the simulations but did not have opportunities to explore in the multi-agent simulation environment. In another study, Dickes and Sengupta (2013) showed that students as young as 4th graders can develop multi-level explanations of population-level dynamics in a predator-prey ecosystem after interacting with a MABM of a bird-butterfly ecosystem through scaffolded learning activities. In our previous work, we identified and quantified the benefits of a set of scaffolds that can help middle school students learn the correct inter-species relations underlying a desert ecosystem simulation, and understand and reason about the concepts of interdependence and balance in such an ecosystem (Basu, Sengupta & Biswas, 2014). Further, Kapur & Kinzer (2008) along with Pathak, et. al. (2008) have shown that learners who are scaffolded and provided structured steps to follow during their ABM activity, initially learn better than unscaffolded learners. However, if this initial activity is followed by a second activity where both groups of learners are scaffolded, and a third activity where both groups are not scaffolded, the initially unscaffolded group will perform better by the third activity. The initial failure of the unscaffolded group in the first study is referred to as 'productive failure', meaning that the initial failure made the students better prepared to learn when they were scaffolded in the second activity.

3.2.3 Summary of existing research on ABMs and ABMs as tools for CT

Review of the literature indicates that the agent-based modeling paradigm has shown great potential for helping students learn emergent phenomena in diverse domains, especially in science domains where a multitude of curricular topics can be modeled as a large, distributed set of agents. Bootstrapping students' intuitive agent-level understandings helps them learn aggregate-level emergent outcomes more easily than when they are presented with aggregate level formalisms directly. However, in order to learn effectively using MABMs, students generally require scaffolds to help them structure their investigations using MABMs, reminders to pay attention to both agent-level and aggregate outcomes, and help understanding the effects of agent-level behaviors on aggregate outcomes.

In addition, we find that agent-based modeling and simulation can serve as powerful tools for introducing CT across the K-12 science curriculum. Modeling each agent type individually as a set of rules that govern its behavior and interactions with other agents encourages the CT practice of decomposing a complex modeling task into manageable pieces, which can be worked on in parallel. ABM also simplifies the process of model debugging and encourages the CT practice of testing effectively by testing individual agents separately. Not surprisingly, we notice that learning environments that try to integrate CT with science learning at the K-12 level tend to use an agent-based modeling paradigm. Learning using such agent-based CT environments would presumably necessitate scaffolding, not just for becoming proficient in CT and science concepts, but also for learning how to work effectively with ABMs. This approach of using ABM as a tool for integrating CT with curricular science topics guides our design and development of the CTSiM learning environment, as described in detail in Section 3.6.

3.3 Visual Programming for middle school students

In a visual programming (VP) environment, students construct programs using graphical objects, typically in a drag-and-drop interface, thus making the programming more intuitive and accessible to the novice programmer (Kelleher & Pausch, 2005; Hundhausen and Brown 2007). Visual constructs significantly reduce issues with learning program syntax and understanding textual structures making it easier for students to focus on the semantic meaning of the constructs (Hohmann, 1992; Soloway, 1993). For example, visual interfaces make it easier to interpret and use flow of control constructs, such as loops and conditionals (Parsons & Haden, 2007). This is an important affordance of VP, because prior research has shown that students in a LOGO programming-based high school physics curriculum faced significant challenges in writing programs for modeling kinematics, even after multiple weeks of programming instruction (Sherin et al. 1993). In the studies reported by Sherin et al. (1993) and diSessa et al. (1991a, b), middle and high school students required fifteen or more weeks of instruction, out of which, the first 5 weeks of classroom instruction were devoted solely to learning programming taught by a programming expert. Sherin et al. (1993) pointed out that, given the time constraints already faced by science (in their case, physics) teachers, the additional overhead associated with teaching students to program may simply prove prohibitive.

Some examples of agent-based VP environments are AgentSheets (Repenning 1993), StarLogo TNG (Klopfer et al. 2009), Scratch (Maloney et al. 2004), ToonTalk (Kahn 1996), Stagecast Creator (Smith et al. 2000), and Alice (Conway 1997). Users in all of all these environments can: (a) construct or design their programs by arranging icons or blocks that represent programming commands and (b) employ animations to represent the enactment (i.e., the execution) of the user-generated algorithm (i.e., program), albeit with varying degrees of algorithm visualization (Hundhausen and Brown 2007).

In the CTSiM learning environment, we focus on VP as the mode of programming and computational modeling to make it easier for middle school students to translate their intuitive knowledge of scientific phenomena (whether correct or incorrect) into executable models that they can then analyze through simulations. CTSiM provides a library of visual constructs that students can choose from and arrange spatially to generate their computational models. If students try to drag and drop a programming construct incorrectly, the system disallows the action, and indicates the error by explicitly displaying an ‘x’ sign. Therefore, CTSiM eliminates the possibility of generating programs (that is, models) with syntax errors.

3.4 Integration of domain-specific primitives and domain-general abstractions

Previous research suggests that learning a domain-general programming language and using it for domain-specific scientific modeling involves a significant pedagogical challenge (Guzdial, 1994; Sherin *et al.,* 1993). Rather, a domain-specific modeling language (DSML) that combines domain-general computational primitives and domain-specific primitives, can help leverage students’ intuitions about the domain, while emphasizing the generality of computational primitives across domains. Domain-general primitives are computational constructs, like “when-do-otherwise do” and “repeat” that represent CT concepts, like conditionals and loops. Domain-specific primitives, on the other hand, are designed specifically to support modeling of particular aspects of the topic of study. Imposing domain-specific names on the constructs creates semantically meaningful structures for modeling actions in the particular domain that help gain a better conceptual understanding of the domain. CTSiM uses a DSML to help foster synergistic science and CT learning. For example, CTSiM uses domain-specific primitives, like “forward”, “speed up” and “slow down” to represent movement, acceleration and deceleration actions in the kinematics domain, and primitives, like “create new” and “die” to imply birth and death of agents in the ecology domain. Students develop complex agent behaviors in CTSiM by meaningfully combining domain-general computational primitives and domain-specific primitives.

3.5 Initial CTSiM architecture and interfaces

Based on the design principles described in Sections 3.1-3.4, we developed the initial version of the CTSiM learning environment, which we will henceforth refer to as CTSiM version 1 (CTSiM v1). CTSiM v1 adopted a learning-by-design pedagogy, where students built computational models of given science topics using an agent-based, visual programming platform and a domain-specific modeling language. Students were also provided support for programming through tools for algorithm visualization and model verification. They could observe the behavior of their computational models as agent-based simulations at any point of time, helping them understand the relationships between their models and the resultant enactment of their models. Students were also provided with an "expert" (i.e., canonically correct) simulation, and could iteratively refine their models by comparing results of their simulation to the expert simulation, understanding the differences, and then mapping behaviors exhibited by their simulations to programming constructs in their computational models and vice-versa.

The initial CTSiM learning environment comprised three primary interface modules: the Construction world (C-World) or the 'Build' interface, where students constructed their computational models; the Enactment world (E-World) or the 'Run' interface for model visualization as simulations; and the Envisionment world (V-World) or the 'Compare' interface for model verification. We describe each of these interfaces in more detail using examples from a curricular unit in ecology implemented using CTSiM v1.

The C-World provided the VP interface for students to build computational models of science topics using an agent-based framework. Figure 1 illustrates the CTSiM v1 C-World drag-and-drop interface that shows a model describing how a fish agent breathes. In this version of the system, students selected the agent and procedure they wanted to model from drop-down menus located at the top of the interface, selected primitives from a palette of programming primitives or blocks placed on the left panel of the interface, and dragged and dropped the selected primitives on the right panel of the interface, spatially arranging and parameterizing them into a structure that represented their computational models for the agent behaviors. For each agent, students modeled the different agent behaviors as separate procedures and specified the procedures to be executed within a procedure called 'Go' for the particular agent. The palette of programming primitives provided for modeling any agent procedure comprised both domain-general computational primitives, like conditionals, loops, variables and operators, as well as domain-specific primitives that represent agents, agent properties, environmental elements, sensing conditions, and agent actions.

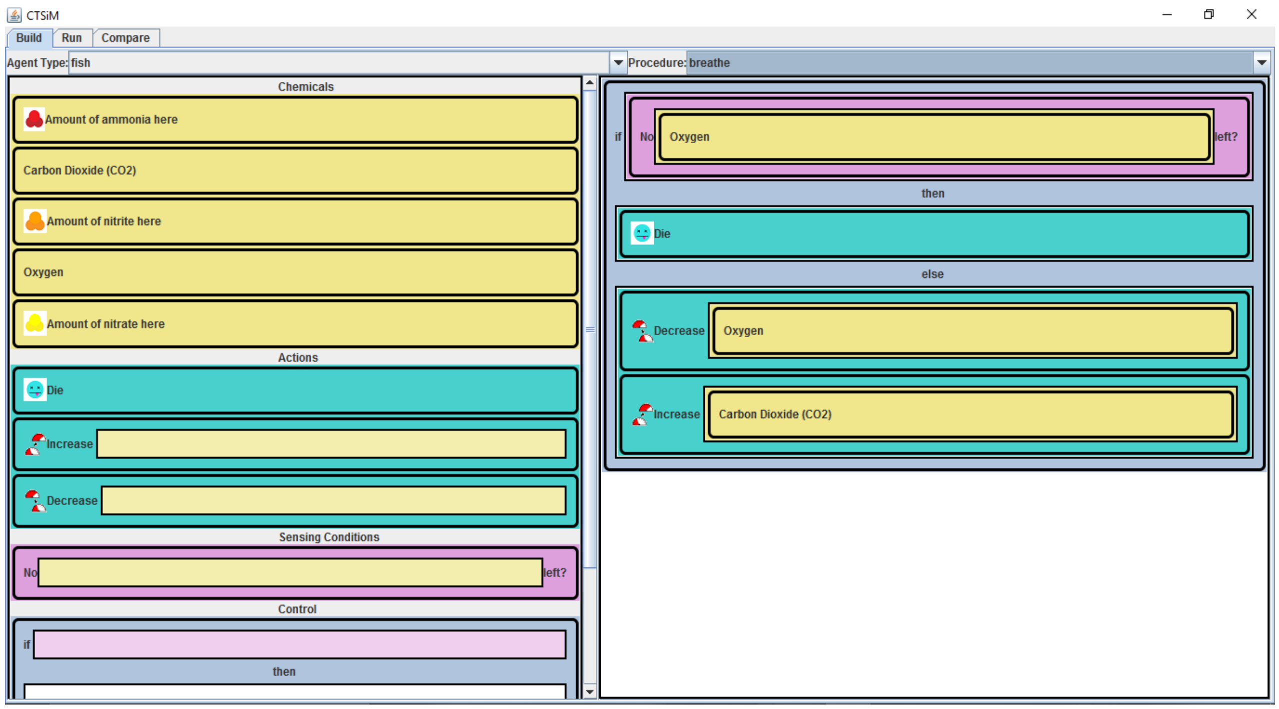


Figure 1. The CTSiM v1 Construction World for building computational models

The E-World represented a microworld (Papert 1980; White and Frederiksen 1990), where the behaviors of agents defined in the C-World could be visualized in a simulation environment. The CTSiM environment, implemented in Java, included an embedded instance of NetLogo (Wilensky 1999) to implement the visualization and mechanics of the simulation. NetLogo visualization and plotting/measurement functionality provided the students with a dynamic, real-time display of how their agents operated in the microworld, thus making explicit the emergence of aggregate system behaviors (e.g., from graphs of the population of a species over time). Students could also view the different agent procedures they modeled alongside the aggregate simulation to better understand the relations between the models they constructed and the resultant simulations. Figure 2 depicts the CTSiM v1 E-World interface.

The visual blocks, or programming primitives used by students as they built their computational models were internally translated to an intermediate language and represented as code graphs of parameterized computational primitives. These code graphs remained hidden from the learner, and were translated into NetLogo code by the model translator. The generated NetLogo code was combined with the domain base model to generate the simulations corresponding to the user models. The base model provided NetLogo code for visualization and other housekeeping aspects of the simulation that were not directly relevant to the learning goals of the unit.

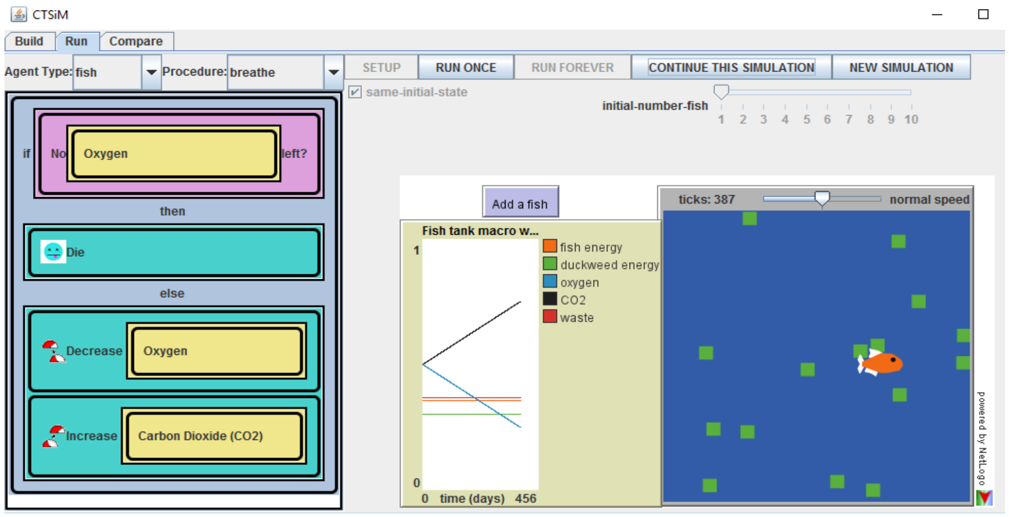


Figure 2. The CTSiM v1 Enactment World for model vizualization

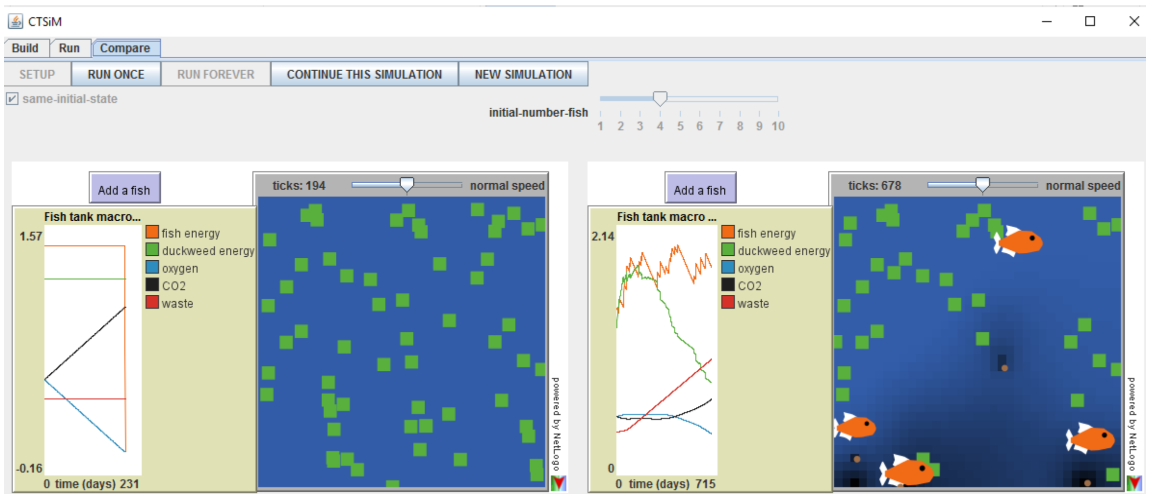


Figure 3. The CTSiM v1 Envisionment World for model verification

The CTSiM v1 V-World provided students with a space where they could systematically design experiments to test their constructed models and compare their model behaviors against behaviors generated by an “expert” model, which ran in lock step with the student-generated model. Although the expert model was hidden, students could observe its generated behaviors and compare them to the corresponding behaviors generated by their model with side-by-side plots and microworld visualizations. This comparison allowed students to make decisions on what components of their models they needed to investigate, develop further, or check to correct for errors. With proper support and scaffolding, we believed that the overall process of model construction, analysis, comparison, and refinement would help students gain a better understanding of science phenomena and the scientific reasoning process, while also learning computational constructs and methods.

3.6 Designing learning activities and a learning progression using CTSiM v1

Kinematics (physics) and ecology (biology) were chosen as the curricular topics for synergistic learning of science and CT using CTSiM v1. They are common and important curricular topics in the middle school curriculum, and researchers have shown that K-12 students have difficulties in understanding and interpreting concepts in these domains (Chi et al. 1994). Furthermore, it has been argued that students' difficulties in both the domains have similar epistemological origins, in that both kinematics phenomena (e.g., change of speed over time in an acceleration field) and system-level behaviors in an ecosystem (e.g., population dynamics) involve understanding aggregation of interactions over time (Reiner et al. 2000; Chi 2005). For example, physics educators have shown that understanding and representing motion as a process of continuous change is challenging for novice learners (Halloun and Hestenes 1985; Elby 2000; Larkin et al. 1980; Leinhardt et al. 1990; McCloskey 1983). Novices tend to describe or explain any speed change(s) in terms of differences or relative size of the change(s), rather than describing speeding up or slowing down as a continuous process (Dykstra & Sweet 2009). Similarly, in the domain of ecology, biology educators have shown that while students have intuitive understandings of individual-level actions and behaviors, they find aggregate-level patterns that involve continuous dynamic processes—such as interdependence between species and population dynamics—challenging to understand without pedagogical support (Chi et al. 1994; Jacobson & Wilensky, 2006; Wilensky & Novak 2010; Dickes & Sengupta, 2012).

Also, as discussed in Section 3.2, agent-based modeling is well suited for representing such phenomena, as it enables learners to invoke their intuitions about agent-level behaviors and organize them through design-based learning activities (Kolodner et al. 2003), in order to explain aggregate-level outcomes. Studies have shown that pedagogical approaches based on agent-based models and modeling can allow novice learners to develop a deep understanding of dynamic, aggregate-level phenomena— both in kinematics and ecological systems by bootstrapping, rather than discarding their agent-level intuitions (Dickes & Sengupta, 2012; Wilensky & Reisman, 2006; Levy & Wilensky, 2008).

We developed a learning activity progression using CTSiM v1 where students worked on Kinematics unit activities first and then proceeded to work on activities associated with the Ecology unit (Sengupta et. al., 2013). The Kinematics unit focused on modeling Newtonian mechanics phenomena, the relations between speed, distance and acceleration, and graphical representations of motion. The Ecology unit, on the other hand, emulated a simplified fish tank environment, and focused on the concepts of dynamic equilibrium and interdependence among species in the ecosystem. In terms of programming concepts, activities in both the domains required understanding and use of common CT concepts like conditionals, loops, variables. However, the Ecology unit activities involved modeling more complex topics and hence required a greater use of CT practices, like decomposition and modularization. In fact, our rationale behind sequencing the two domains in the curriculum was guided by the programming complexities involved in modeling phenomena in the two domains (Sengupta et. al., 2013). For example, while the kinematics learning activities described below required the students to program the behavior of a single computational agent, modeling the fish tank ecosystem required students to program the behaviors of and interactions between multiple agents. We introduced students to single-agent programming before introducing them to multi-agent programming—therefore, in our curricular sequence, students learnt kinematics first, and then ecology.

For the kinematics unit, we extended previous research by Sengupta & Farris (2012) to design the learning activities in three phases.

Kinematics Phase 1: This covered Activities 1 and 2, where students used *Turtle graphics to construct geometric shapes that represented:* (1) *constant speed and* (2) *constant acceleration.* In Activity 1, students were introduced to programming primitives such as "forward", "right turn", and "left turn" that dealt with the kinematics of motion, primitives like "repeat", which corresponded to a computational construct (independent of a domain construct), and primitives like "pen down", and "pen up," which were Netlogo-specific drawing primitives. The students were given the task of generating procedures that described the movement of a turtle for drawing *n*-sided regular shapes, such as squares and hexagons. Each segment of the regular shape was walked by the turtle in unit time indicating constant speed. Therefore, Activity 1 focused on students learning the relationship between speed, time, and distance for constant-speed motion. In

Activity 2, students were given the task of extending the turtle behavior to generate shapes that represented increasing and decreasing spirals. In this unit, segments walked by the turtle, i.e., its speed per unit time, increased (or decreased) by a constant amount, which represented a positive (or negative) acceleration. Activity 2, thus introduced students to the relations between acceleration, speed, and distance using the “speed up” and “slow down” commands to command the motion of the turtle.

Kinematics Phase 2 corresponded to Activity 3, where students *interpreted a speed-time graph to construct a representative turtle trajectory.* Starting from the speed-time graph shown in Figure 4 students developed a procedure where the length of segments the turtle traveled during a time interval corresponded to the speed value on the graph for that time interval. For example, it was expected that students would recognize and model the initial segment of increasing speed by a growing spiral, followed by the decrease in speed by a shrinking spiral, whose initial segment length equaled the final segment length of the last spiral. Students were given the freedom to choose the shapes associated with the increasing and decreasing spirals. A possible solution output is provided in Figure 4. We hypothesized this reverse engineering problem would help students gain a deeper understanding of the relations between acceleration, speed, distance, and time.

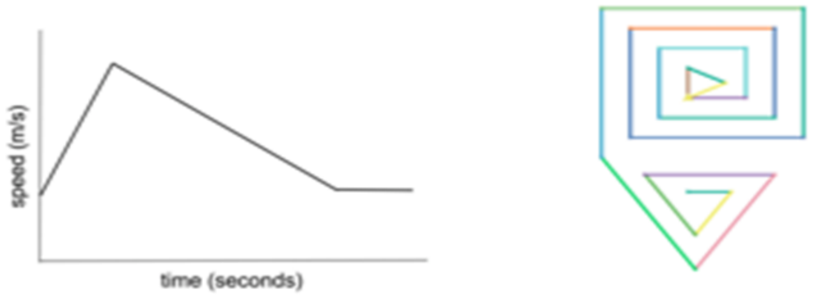


Figure 4. Acceleration represented in a speed-time graph (left) and turtle graphics (right)

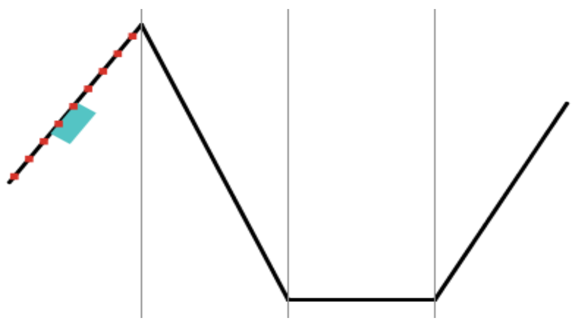


Figure 5. A roller-coaster car moving along different segments of a pre-specified track

Kinematics Phase 3, represented by Activity 4 involved *modeling the motion of a rollercoaster car along a pre-specified track with multiple segments* (see Figure 5). In more detail, students were asked to model a rollercoaster as it moved through different segments of a track: (1) up (pulled by a motor) at constant speed, (2) down (with gravitational pull), (3) flat (cruising), and then (4) up again (moving against gravity). The students had to build their own model of rollercoaster behavior to match the observed expert behavior for all of the segments.

For the Ecology unit, students modeled a closed fish tank system in two phases, represented by Activities 5, 6, and 7.

Ecology Phase 1 corresponded to Activity 5, where students constructed a macro-level, semi-stable model of the fish tank ecosystem by modeling the fish and duckweed species as two agent types (see Figures 2 and 3). Activity 5 required students to model the food chain, respiration, locomotion, excretion, and the reproductive behaviors for the fish and duckweed. The inability to develop a sustained macro-model, where the fish and the duckweed could be kept alive for extended periods of time, even though all of the macro processes associated with the two agents were correctly modeled (that is, the behaviors generated by the students' computational model matched the behaviors generated by the expert model), encouraged students to reflect on what may be missing from the macro model. Students realized the need to model the *waste cycle* and its entities, primarily the two forms of bacteria and their behaviors. This prompted the transition to the second phase where students identified the continuously increasing fish waste as the culprit for the lack of sustainability of the fish tank.

Ecology Phase 2, represented by Activity 6, involved building the waste cycle model for the fish tank, with the Nitrosomonas bacteria that converts the toxic ammonia in fish waste into nitrites, which is also toxic, and the Nitrobacter bacteria that converts the nitrites into nitrates. Nitrates are consumed by the duckweed (as nutrients), thus preventing an excessive buildup of toxic chemicals in the fish tank environment. The combination of graphs from the micro- and macro-world visualizations was intended to help the students develop an aggregate-level understanding of the interdependence and balance among the different agents (fish, duckweed, and bacteria) in the system. After completing the ecology micro unit, students worked on Activity 7, where they discussed the combined micro-macro model with their assigned researcher and how the macro-micro model phenomena could be combined into an aggregated causal model describing the sustainability of the fish tank ecosystem.

CHAPTER 4

Assessing students' use of the initial version of the CTSiM learning environment

We conducted a semi-clinical interview-based study (Piaget, 1929) using the initial version of CTSiM described in Chapter 3 to assess the effectiveness of our pedagogical approach and study students' interactions with the system (Basu et. al., 2013; Sengupta et. al., 2013). We wanted to see if students could model science topics and learn using CTSiM v1, and study what problems and questions they had while working using the system. This chapter lays out the particulars of this initial study and our findings which influence future re-design of the CTSiM environment.

4.1 Research study setting and procedure

We conducted an initial study using CTSiM v1 with one 6th-grade classroom comprising 24 students (age ranges between 11 and 13) from an ethnically diverse middle school in central Tennessee in the United States. Since this was our first study with the CTSiM v1 system, our primary goals were to gain a detailed understanding of how students interacted with the system, the approaches they used in constructing their models, the problems they faced while building and debugging their models, how they discovered and responded to errors in their models, and scaffolds that could help them deal with their challenges and complete their modeling activities. Towards this end, 15 of the 24 students in the class were chosen by the science teacher to work on CTSiM v1 with one-on-one individualized verbal guidance from members of our research team. The teacher ensured that the chosen students were representative of different genders, ethnicities, and performance levels. We refer to these students as the Scaffolded or S-Group. While the majority of the class participated in the pull-out study, the remainder of the class (9 students) was allowed to explore the same set of units in the CTSiM v1 environment on their own without the one-on-one guidance. However, these students did get some guidance from the teacher and members of our research team during their science period. We refer to these nine students as the Classroom or C-Group.

The 15 students in the S-Group were paired one-on-one with one of five members of our research team. Thus, each researcher from our team worked with 3 students for the study with three 1-hour sessions daily (9am-10am, 10am-11am, 12:30pm-1:30pm), one for each student assigned to them. All five members of our research team who conducted the one-on-one interviews had prior experience with running similar studies. The members met before the study and decided on a common framework for questioning and interacting with students as students worked with CTSiM v1. While the interviews were not strictly scripted, since the conversations would depend on individual student actions and thought processes, a common flexible interview script was prepared and shared among the researchers. The interview script ensured that all of the researchers' interview formats and structures were similar (similar questions asked and similar examples to illustrate a concept) during each of the CTSiM learning activities. As part of the intervention, the researchers introduced the CTSiM v1 system and its features to their students individually, and introduced each of the learning activities before the student started them. However, students were not told what to do; they had complete control over how they would go about their modeling and debugging tasks. But the researchers did intervene to help students when they were stuck or frustrated by their own lack of progress. An important component of the researchers' interactions with the students involved targeted prompts, where they got students to focus on specific parts of the simulation results and verify the correctness of their model. When needed, the researchers also asked leading questions to direct the students to look for differences between the expert simulation results and their own results, and then reflect on possible causes for observed differences. These questions often required students to predict the outcome of changes they had made to their models and then check if their predictions were supported by the simulation results. In addition, the researchers prompted the students periodically to make them think aloud and explain what they were currently doing on the system. They also provided pointers about how to decompose large complex modeling problems into smaller manageable parts, and at appropriate times, reminded students about how students had tackled similar situations in past work. All of the student and researcher conversations during the one-on-one interviews were recorded using the Camtasia software. These videos also included recordings of the screen, so we could determine what actions the students performed in the environment, and what the consequences of the actions were.

Since this study was primarily targeted toward understanding how students' used the system, and how their learning and understanding of the science processes evolved, we only assessed science learning as a result of our intervention and did not include assessments for CT. The science assessments included paper-based kinematics and ecology questions (the pre- and post-tests design included repeated items), which comprised a combination of multiple choice and short answer questions. On Day 1 of the study, we administered pre-tests for both the Kinematics and Ecology units. Students took between 25 and 40 minutes to finish each test. Then students worked on the kinematics units (Activities 1-4) in daily hour-long sessions from Day 2 through Day 4 and took the kinematics post-test on Day 5. On Days 6-8, students worked in daily hour-long sessions on the ecology units (Activities 5-7) and then took the ecology posttest on Day 9. The entire study took place over a span of two weeks towards the end of the school year, after the students had completed their annual state-level assessments (Tennessee Comprehensive Assessment Program or TCAP).

The Kinematics pre/post-tests were designed to assess students' understanding of concepts like speed, distance, and acceleration and their relations, along with their reasoning using mathematical representations of motion (e.g., graphs). Our goal was to assess whether computational modeling improved their abilities to generate and explain these representations. Specifically, students were asked to interpret and explain speed versus time graphs, and to generate diagrammatic representations to explain motion in a constant acceleration field, such as gravity. For example, one question asked students to diagrammatically represent the time trajectory of a ball dropped from the same height on the earth and the moon. The students were asked to explain their drawings and generate graphs of speed versus time for the two scenarios.

For the Ecology unit, the pre- and post-tests focused on students' understanding of roles of species in the fish-tank ecosystem, interdependence among the species, the food chain, the waste and respiration cycles, and how a specific change in one species affected the others. Some of the questions required students to use declarative knowledge about the fish tank system while other questions required causal reasoning about entities using the declarative knowledge. An example question asked was "*Your fish tank is currently healthy and in a stable state. Now, you decide to remove all traces of nitrobacter bacteria from your fish tank. Would this affect a) Duckweed, b) Goldfish, c) Nitrosomonas bacteria? Explain your answers.*"

4.2 Science learning gains using CTSiM

We assessed students' science learning in this initial study based on their kinematics and ecology pre-to-post learning gains. Two members of our research team (including myself) came up with initial rubrics for grading the tests, which were then iteratively refined based on student responses.

The initial rubric focused on correct answers for multiple choice questions and keywords and important concepts for questions requiring short answer responses. A systematic grading scheme was developed after studying a subset of the student responses. The short answer grading scheme attempted to account for different ways a question could be answered correctly, and was updated if we found a student response which could not be graded adequately using the current rubric.

Students belonging to both the S and C groups showed learning gains in both the curricular units (Basu et. al., 2012; Sengupta et. al., 2012). The pre-test scores and the state-level TCAP science scores from the previous academic year suggested differences in prior knowledge and abilities of students in the S and C groups. Hence, we computed repeated measures ANCOVA with TCAP science scores as a covariate of the pretest scores to study the interaction between time and condition. Not surprisingly, there was a significant effect of condition (i.e., S-Group versus C-Group) on pre-post learning gains in kinematics (F(1,21) = 4.101, p<0.06), as well as ecology (F(1,21) = 37.012, p<0.001). Figure 6 shows that the S group's adjusted gains were higher than that of the C group in both the curricular units.

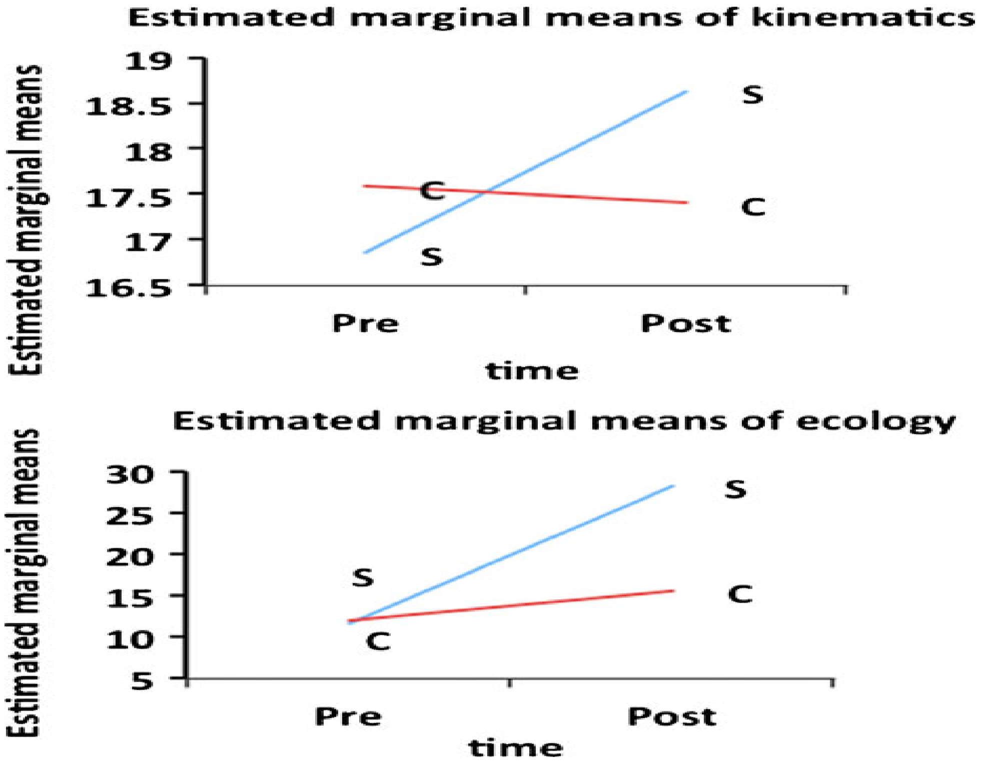


Figure 6. Comparison of learning gains between groups using TCAP scores as a covariate in the first CTSiM study

We also conducted paired sample t-tests to study pre-to-post gains for each group. Table 2 shows that for the ecology unit, the intervention produced statistically significant pre-to-post gains for both groups ($p<0.001$ for S-group; $p<0.01$ for C-Group), but for the kinematics unit, the gains were significant only for the S-Group ($p<0.05$ for S-group, $p>0.5$ for C-group). The reduction in statistical significance for the kinematics unit may be attributed to a ceiling effect in students' pre-test scores, given that students entered the instructional setting with a significantly higher score in the kinematics pre-test than the ecology pre-test.

Table 2. Paired t-test results comparing Kinematics and Ecology pre and post test scores from the first CTSiM study

	Kinematics				**Ecology**			
	Pre Mean (S.D.) (max=24)	**Post Mean (S.D.) (max=24)**	**P-value (2-tailed)**	**Effect size (Cohen's d)**	**Pre Mean (S.D.) (max=35.5)**	**Post Mean (S.D.) (max=35.5)**	**P-value (2-tailed)**	**Effect size (Cohen's d)**
S-Group (n=15)	18.07 (2.1)	19.6 (2.3)	<0.05	0.71	13.03(5.4)	29.4(5.0)	<0.001	3.16
C-Group (n=9)	15.56 (4.1)	15.78 (4.4)	>0.5	0.05	9.61(3.1)	13.78(4.4)	<0.01	1.09

4.3 Identifying students' difficulties while working with CTSiM

Besides assessing students' science learning through their performance on pre- and post- tests, we also analyzed the Camtasia-generated videos for all fifteen students of the S-group to characterize the types of challenges students faced when working with CTSiM and the scaffolds that were provided to help them overcome these challenges.

4.3.1 Analysis and coding of study data

The video data was coded along two dimensions: first, the type and frequency of challenges faced during each activity, and second, the scaffolds that were used to help the students overcome the challenges. Initial codes were established using the *constant comparison* method by two researchers involved in the study (including myself). To do so, we chose data from two participants, whom we will call Sara and Jim (not their real names). Sara and Jim were selected as representative cases, because they had the lowest and highest state standardized assessment (TCAP) scores in science among the 15 participants of the pull-out study. Fourteen challenge categories were identified, which we further grouped into four broad categories: (1) *Programming Challenges,* (2) *Modeling Challenges,* (3) *Domain Challenges*, and (4) *Agent-Based Reasoning Challenges* – to aid in the interpretation of the aggregate data set. Henceforth, we refer to the 14 initial categories as 'sub-categories' of these four broad categories. Definitions and examples of the different types of challenges are explained in detail in Section 4.3.2.

To establish reliability, two researchers unaffiliated with the study coded all the interviews independently, using the described coding scheme. To determine inter-rater reliability, the researchers were first asked to determine the challenges and frequency counts for Activity 3, 4, and 5 from Sara's video data. Both coders reached good agreement with the researcher-developed codes (91.15% and 96.46% agreement). Once reliability with the researcher codes was established, the coders were asked to code a different student to test their inter-rater reliability. The inter-rater reliability between Coder 1 and Coder 2 yielded a Cohen's Kappa of 0.895 (93.1% agreement), implying a 'very good' inter-rater reliability rating. Then, the coders divided up the work of coding the remaining 12 student videos. Once the challenges faced and scaffolds received for all 15 students were extracted from the video files, we computed the average number of challenges of each type per activity in order to better understand the relations between different types of challenges and how the challenges varied across learning activities spanning two domains.

4.3.2 Challenges faced and scaffolds required

Our analysis of the one-on-one interviews produced four primary categories and 14 subcategories of challenges students faced when developing and testing their models using CTSiM v1 (Basu et. al, 2013; Basu et.al., in review), which are summarized as follows:

a. *Domain knowledge challenges* related to difficulties attributed to missing or incorrect domain knowledge in science. Given that these challenges were non-computational in nature, they were not studied in further detail.
b. *Modeling and simulation challenges* were associated with representing scientific concepts and processes as computational models, and refining constructed models (partial or full)

based on observed simulations. More specifically, these challenges included difficulties in identifying the relevant entities in the phenomenon being modeled; specifying how the entities interact; choosing correct preconditions and initial conditions, model parameters, and boundary conditions; understanding dependencies between different parts of the model and their effect on the overall behavior; and verifying model correctness by comparing its behavior with that of an expert model. Subcategories of these challenges could be classified as: (1) challenges in identifying relevant entities and their interactions; (2) challenges in choosing correct preconditions; (3) systematicity challenges; (4) challenges with specifying model parameters and component behaviors; and (5) model verification challenges).

c. *Agent-based thinking challenges* – Students faced difficulties in expressing agent behaviors as computational models, difficulties in understanding how individual agent interactions lead to aggregate-level behaviors and the consequences of agent behavior changes on the aggregate behavior. Therefore, the subcategories of challenges have been called: (1) thinking like an agent challenges; and (2) agent-aggregate relationship challenges.

d. *Programming challenges* – Students had difficulties in understanding the meaning and use of computational constructs and other visual primitives (for example, variables, conditionals, and loops). They had difficulties in conceptualizing agent behaviors as distinct procedures, and some could not figure out how to compose constructs visually to define an agent behavior. Additional difficulties were linked to the inability to reuse code, and to methodically detect incorrect agent behavior, find root causes, and then figure out how to correct them. The programming challenge subcategories were: (1) challenges in understanding the semantics of domain-specific primitives; (2) challenges in using computational primitives like variables, conditionals, nesting, and loops to build programs (i.e., behaviors); (3) procedurality challenges; (4) modularity challenges; (5) code reuse challenges; and (6) debugging challenges).

These four types of challenges are not mutually exclusive. For example, agent-based thinking challenges could also be considered as modeling and simulation challenges, but specific to the agent-based modeling paradigm we have employed in CTSiM. However, this categorization still offers ease of analysis and reporting. Tables 3, 4, 5 and 6 describe the domain knowledge, modeling, agent-based-thinking, and programming challenges respectively, along with subcategories of challenges where applicable, examples of occurrence of the challenges from the kinematics and ecology units, and scaffolds provided by the experimenters to help students overcome these challenges.

Table 3. Domain knowledge challenges and scaffolds

Challenge	Description	Kinematics unit examples	Ecology unit examples	Scaffolds provided
Domain knowledge related challenges	Difficulties caused by missing or incorrect domain knowledge	Difficulty understanding acceleration and its relation to speed, how speed depends on the rollercoaster segment slope	Lack of prior knowledge about the waste cycle in the fish tank, the chemicals and the role of bacteria	Explain formal procedures for calculations, provides definitions, explanations, and examples of different scientific terms and concepts; Help connect domain-related theoretical concepts to learning tasks in the CTSiM environment; Rectify incorrect knowledge using contrasting cases for creating cognitive conflict

Table 4. Types of modeling and simulation challenges and scaffolds.

Types of challenges	Description	Kinematics unit examples	Ecology unit examples	Scaffolds provided
Challenges with identifying relevant entities and their interactions	Difficulty identifying the agents, their properties and their behaviors; which properties a behavior depends on and which properties a behavior affects, and how different agents interact with each other	Modeling work done and energy consumed instead of speed of the roller coaster; Difficulty understanding relation between steepness and speed	Difficulty identifying types of environmental components (in this cases, gases) that are needed to model procedures like 'breathe' and 'eat'	Interviewer points out the aspects of the phenomena that need to be modeled; Interviewer prompts students to think about the agents to be modeled, their properties and behaviors, and the interactions between agents, and agents and their environment
Challenges with choosing correct preconditions	Difficulty in identifying and setting appropriate initial conditions and preconditions for different processes and actions	Difficulty understanding that modeling acceleration requires specifying an initial velocity	Difficulty understanding that a fish needs to be hungry and needs to have duckweed	Prompt students to think about the preconditions necessary for certain functions/behaviors; Encourage students to

			present to be able to eat	vary initial conditions and test outcomes
Systematicity challenges	Difficulty in methodical exploration; Guessing and modifying the code arbitrarily instead of using the output behaviors to inform changes	Non-systematic exploration and testing of different turn angles to generate a triangle or circle	Lack of confidence about model being built; Changing model arbitrarily in an attempt to correct errors	Encourage students to think about their goals, the starting points, and their plans of action
Challenges with specifying model parameters and component behaviors	Difficulty determining parameters for the visual primitive blocks in the C-World to produce measurable and observable outcomes, and understanding individual effects of different components of a code segment on the behavior of the entire code segment	Difficulty choosing optimal input parameters to generate clearly visible outputs; Confusion understanding effects of turn angle, speed up factor, and number of repeats on figure dimensions	Inability to specify outcomes when a condition is true and when it is not, for example a fish dies when there is no oxygen	Prompt students to make changes in parameter values to produce clearly visible outputs; Encourage testing outcomes by varying parameter values
Model verification challenges	Difficulty verifying and validating the model by comparing its behavior with that of the given expert model and identifying differences between the models	Difficulty comparing user and expert rollercoaster models; Difficulty correlating model with simulation	Difficulty comparing user and expert fish tank models; Difficulty correlating changes in the model and changes in user model output	Ask students to slow down the simulation to make agent actions more visible; Point out the differences between the user and expert models

Table 5. Types of Agent-based thinking challenges and scaffolds

Types of challenges	Description	Kinematics unit examples	Ecology unit examples	Scaffolds provided
Thinking like an	Difficulty in modeling a phenomenon in terms of one or	Problem delinking turn	Difficulty modeling	Drawing on paper and explaining; Making the

agent challenges	more agents, their properties and their associated sets of distinct rules	angle and forward movement to generate shapes; Difficulty understanding effects of turning with respect to different headings	how an agent gains and loses energy; Problem delinking related actions – 'face nearest' does not mean going forward as well	students imagine themselves as agents; Providing external tools and artifacts to help understand and replicate agent behavior; Enacting agent behavior and making students predict such behavior; Prompts to visualize agent behavior mentally; Reminder that an agent does only what it is programmed to do
Agent-aggregate relationship challenges	Difficulty understanding that aggregate level outcomes can be dependent on multiple agent procedures and debugging such a procedure requires checking each of the associated agent procedures; Difficulty reasoning about the role and importance of individual agents in an aggregate system	Did not occur	Difficult understanding that aggregate outcomes like o2 levels may depend on multiple agent procedures	Reminder about different agents which can affect a particular aggregate level outcome

Table 6. Types of programming challenges and scaffolds

Types of challenges	Description	Kinematics unit examples	Ecology unit examples	Scaffolds provided
Challenges with semantics and execution of domain-specific primitives	Difficulty understanding the functionality and role of various visual primitives and their execution semantics	Difficulty understanding how 'right_', 'speed up' blocks work and how to use them correctly	Did not occur	Step through the code and explain the functionality of primitives by showing their behavior in the E-World; Explain correct syntax for primitives
Challenges with computational	Difficultly in understanding the concept of variables, iterative-structures or	Difficulty coordinating loops and turn	Difficulty with conditionals and nesting conditionals	Explain concept of a variable using examples; Explain

primitives like variables, conditionals, nesting, and loops	loops, conditionals and how and when to nest conditionals within other conditional statements	angles to generate shapes, understanding what it means to increase the speed by the 'steepness' variable	to represent multiple preconditions which needed to be satisfied simultaneously	syntax and semantics of loops and nested conditions using code snippets and their enactment
Procedurality challenges	Difficulty specifying a modeling task as a finite set of distinct steps, and ordering the steps correctly to model a desired behavior	Did not occur	Difficulty specifying behaviors like eat, breathe as a computational structure made up of a small set of primitive elements	Prompt students to describe the phenomena, and break the phenomena into subparts and the individual steps within each subpart
Code reuse challenges	Difficulty identifying already written similar code to reuse and understanding which parts of the similar code to keep intact and which to modify	Did not occur	Difficulty understanding that 'breathe' procedures for Nitrosomonas and Nitrobacter bacteria are similar and can be reused	Prompts encouraging analogous reasoning; Making students think about what similar procedures they have already written
Modularity challenges	Difficulty in separating the behavior of the agents into independent procedures such that each procedure executes only one functionality or aspect of the desired agent behavior, independent of other functionalities in other procedures, along with difficulty remembering to call/invoke each of the procedures from the main procedure or program	Did not occur	Difficulty modeling the fish 'eat' and 'swim' behaviors separately in different procedures (Though eating and swimming together is possible in real life, modeling calls for distinct procedures); Forgetting to call procedures from the main 'Go' method	Prompt students to think about which function/behavior they are currently modeling and whether their code pertains to only that function

Debugging challenges	Difficulty in methodically identifying 'bugs' or unexpected outcomes in the program, determining their underlying causes, removing the bugs and testing to verify the removal of the bugs	Difficulty testing and correcting behavior of one rollercoaster segment at a time	Did not occur	Prompt students to walk through their codes and think about which part of their code might be responsible for the bug; Help break down the task by trying to get one code segment to work before moving onto another

4.3.3 Number of challenges and their evolution across activities

As further analysis beyond the different types of difficulties students faced when working with CTSiM, and the scaffolds which helped them in such situations, we also studied how the frequency of challenges varied across learning activities in one domain and across domains. This analysis helped understand the complexities associated with different learning activities and the variation in support required in these activities.

First, we ran an agglomerative complete-link hierarchical clustering algorithm to see how the students grouped based on their challenge frequency profiles per activity. The results showed that the students generally exhibited similar challenge profiles with the exception of one student. Figure 7 shows the challenge profiles of the two clusters – the average challenge profile for the similar group of 14 students, and one outlier, a single student who seemed to face many more challenges than the rest of the students. This student needed more scaffolding than the other students, and several challenges had to be scaffolded more than once before the student could overcome those difficulties. This student's pre-test and standardized state-level test scores were much lower than that of the other students, which may explain why the student had a significantly higher number of challenges initially. Though this student had multiple challenges that persisted through multiple activities, the number of challenges the student faced came closer to the number of challenges the others faced at the end of the kinematics (Activity 4) and ecology units (Activity 7). Similarly, the student's post-test scores also matched that of the others, making this student's pre-post gains higher than most of the students.

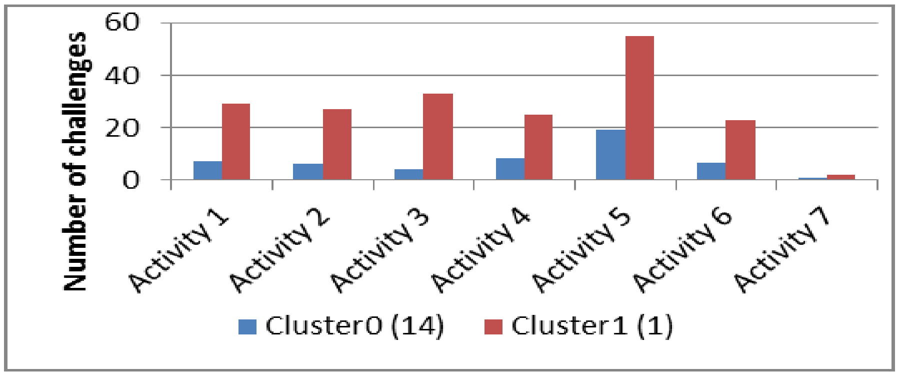


Figure 7. Students clustered according to their number of challenges per activity in the initial CTSiM study

Next, we analyzed how the average number of challenges per student varied across the kinematics and ecology units and across the activities in each unit. The average number of challenges for an activity is calculated as the total number of challenges for all 15 students for an activity divided by 15. This number depends on new challenges that students face in an activity, as well as the effectiveness of scaffolds received in previous activities. Whenever students faced challenges in an activity, the researchers provided scaffolds through conversations to help them overcome their challenges. If the scaffolding was successful, students would be more likely to overcome future occurrences of these challenges in their model building and checking tasks. However, we did observe students encountering similar challenges later in the same activity or in subsequent activities, and, therefore, students received the same or similar scaffolding more than once. Latter conversations associated with scaffolds often started with a reminder that the scaffold had been provided earlier when the student faced the same challenge.

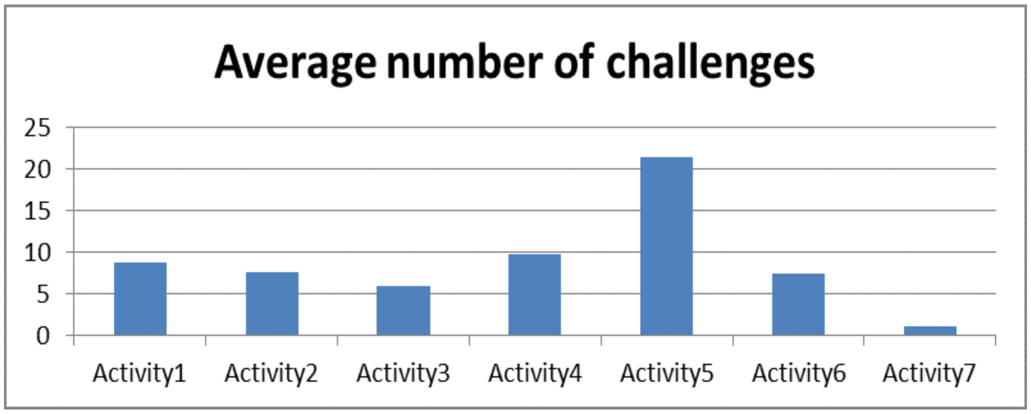


Figure 8. Variation of average number of challenges over activities in the initial CTSiM study

Figure 8 shows how the average number of challenges varied across the different activities. The number of challenges generally decreased across similar activities in the same domain. For example, the number of challenges decreased through the progression of shape drawing activities (Activities 1-3); similarly, the challenges decreased from Activity 5 through Activity 7 for the ecology units. The challenges increased when there was a transition from one domain to another (Activity 4 in kinematics to Activity 5 in ecology) and between problem types in a domain (Activity 3 to Activity 4 in the Kinematics domain). The increase in challenges was because Activity 4 (the rollercoaster activity) introduced a number of new modeling and programming challenges that the students had not encountered in the previous units. The complexities of modeling a real world phenomenon required that the students account for all of the relevant variables, such as the steepness of the roller-coaster ramp. In addition, this was the first activity where the students' simulation model behaviors had to match that of an expert model behavior. Students required a better understanding of the simulation output, which was presented as a combination of an animation and graphs. Moreover, this activity was more challenging from a computational modeling viewpoint, because model building required the use of nested conditionals and variables. Students were experiencing these computational concepts for the first time, and this explained the increase in the difficulties they faced. Similarly, when students progressed from the Kinematics domain to the Ecology domain, Activity 5 (the fish-tank macro model) introduced additional complexities attributed to the change in domain. For example, students had to scale up from a single-agent to a multi-agent model. Activity 5 also involved modeling multiple behaviors for each agent, and students had to figure out how to modularize behaviors, for example, what to include in the fish 'eat' behavior versus the fish 'swim' behavior. (The two are related – a fish has to swim to its food before it can eat the food).

The average number of challenges students faced in an activity is, thus a function of the complexity of the activity and the effectiveness of the scaffolds received in previous activities. Since we found an increase in average number of challenges in Activities 4 and 5, we further reviewed the coded student videos to analyze whether the challenges were new ones related to the new complexities introduced in the activities, or whether they were old ones resurfacing despite previous scaffolding. Our analysis showed that a number of new challenges were introduced in Activities 4 and 5, though a few previously observed challenges also resurfaced in the context of the more complex activities. For Activity 4 (RC activity), students faced several new challenges in:

- *Modeling* – Difficulties in comparing user and expert models, difficulties in setting preconditions and initial conditions, and modeling aspects that did not need to be modeled;
- *Programming* – New challenges included difficulties in understanding the concept of ‘variables’, difficulties in understanding the semantics of conditionals and nesting of conditionals, difficulties in debugging and testing the code in parts;
- *Domain knowledge* - difficulties included understanding that speed varies based on angle of the roller coaster track segment, and difficulties in understanding how rollercoaster motion can be characterized by acceleration and speed.

Similarly, the increase in challenges from Activity 4 to Activity 5 can be attributed to a set of new challenges in:

- *Programming* – difficulties covered the inability to decompose behaviors into separate procedures, define procedures, but forget to call them from the ‘Go’ procedure, and challenges in decomposing a behavior into a sequence of steps;
- *Domain* – difficulties included missing or incorrect knowledge about what duckweed feed on, and what increases and decreases fish and duckweed energy;
- *Agent-based thinking* – difficulties in understanding energy states of agents, difficulties in understanding that aggregate outcomes may depend on multiple agent procedures.

Next, we looked at previously observed challenge categories which resurfaced and increased in Activities 4 and 5. In Activity 4, the only previously observed challenges that increased with time were the programming challenge related to understanding the syntax and semantics of domain specific primitives, and the modeling challenge related to model validation. Facing challenges with respect to understanding domain specific primitives seems understandable in the wake of new domain knowledge and related domain knowledge challenges. Also, Activity 4 marked the first time students had to perform model validation by comparing their model simulations against expert simulations, and had to compare the two sets of animations and plots to assess the correctness of their models. Similarly, in Activity 5, there were a few challenges previously observed in Activity 4 which resurfaced and increased. For example, programming challenges related to use of CT primitives increased, as did modeling challenges related to identifying relevant entities and their interactions, choosing correct preconditions, and specifying model parameters and component behaviors. A new domain, increase in domain complexity, and dealing with modeling multiple agents and multiple behaviors for each agent seem to have been the primary contributors. Further, the size (number of blocks contained) of the fish-macro expert model was about thrice that of the expert roller-coaster model, increasing the probability of facing various difficulties in this activity (Activity 5). Challenges with using CT constructs, like conditionals resurfacing in the context of complex domain content emphasize the need for further practice and a more holistic understanding of the constructs. Unfortunately, we did not study computational learning gains using pre- and post-tests in this initial study, but they may have indicated that students needed repeated practice in different contexts to gain a deep understanding of the computational constructs. Using Figures 8-11, we investigate these issues further, by analyzing the data available from this study to study how the four primary categories of challenges individually varied across activities.

Figure 9 shows that students generally had fewer difficulties with domain knowledge in kinematics (Activities 1-4), than in ecology (Activities 5-7). For kinematics activities, the challenges did increase with the introduction of new domain-specific concepts, like acceleration and the operation of a roller-coaster. But there was a sharp increase in the number of challenges when students had to deal with multiple agents and their interactions in the macro and micro fish tank activities. The difficulties were further compounded by students' low prior knowledge in ecology as indicated by their low ecology pre-test scores.

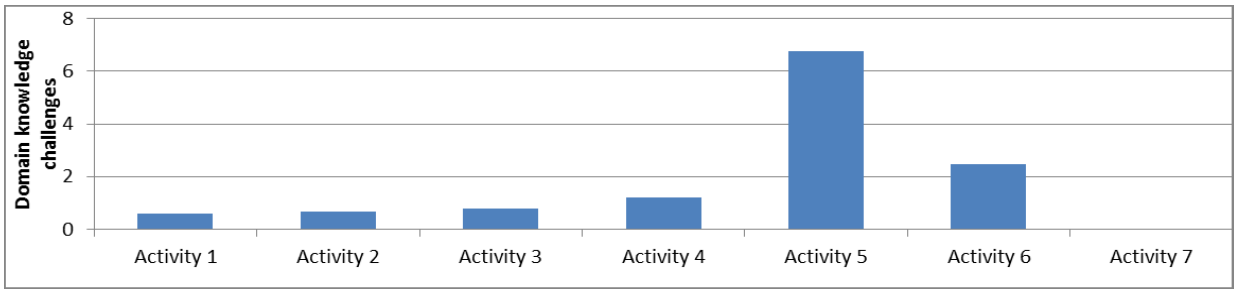


Figure 9. Average number of domain knowledge challenges over time

Programming challenges show a similar trend as seen for average number of challenges in general in Figure 8. Figure 10 shows that students initially had problems with understanding computational primitives, such as conditionals, loops, nesting, and variables, but these programming challenges decreased from Activity 1-3. Activity 4 introduced a new type of programming challenge related to checking and debugging one's model using the results from an expert simulation. Also, challenges with understanding primitives increased due to the number of new primitives (domain-based and computational) introduced in Activity 4. Another big challenge in Activity 4 was constructing nested conditionals to model roller coaster behavior on different segments of the track. In Activity 5, there were new types of programming challenges related to modularity and procedurality, since the fish tank macro-model required students to specify component behaviors as separate procedures that were invoked from one main "Go" procedure. However, challenges with understanding conditionals, loops, nesting, and variables also increased, though these concepts were not new to this activity. The reason for the resurfacing of old challenges may be explained by the increase in the complexity of the domain content in this activity (see Figure 9), making it harder for the students to translate the domain content into computational structures. Overall, for both kinematics and ecology units, the programming challenges decreased over time across activities in the unit unless an activity introduced addition complexities.

Similarly, modeling challenges (see Figure 11) increase in number in Activity 4 for kinematics and Activity 5 for ecology. Initial difficulties were related to systematicity, specifying component behaviors, identifying entities and interactions, and model validation. In Activity 4, modeling a real-world system introduced new challenges related to choosing correct initial conditions. Students also had the additional task of verifying the correctness of their models by comparing against expert simulation behaviors. For Activity 5, although the average number of challenges increased there were no new types of modeling challenges. Existing modeling challenges resurfaced in light of the sharp increase in domain-knowledge related challenges. However, when students switched to the fish-tank micro-unit (Activity 6), they had overcome most of these challenges.

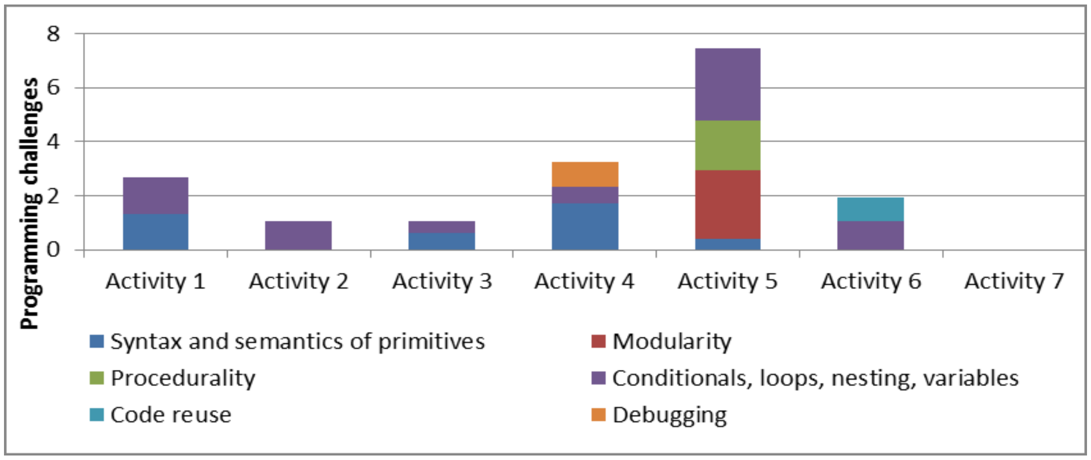


Figure 10. Average number and type of programming challenges over time

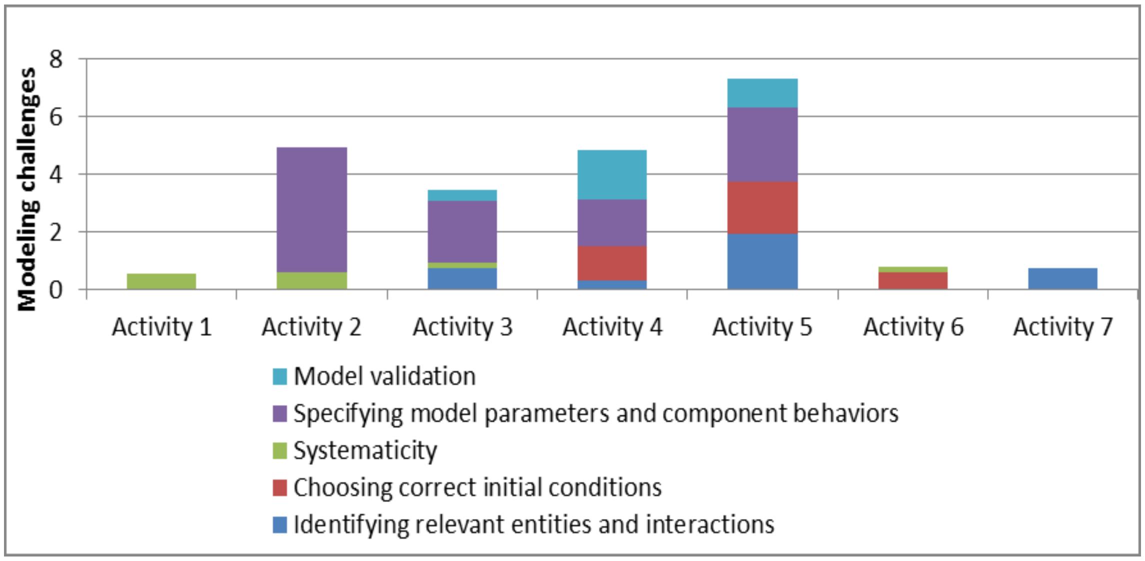


Figure 11. Average number and type of modeling challenges over time

For the agent-based thinking challenges (see Figure 12), challenges went down with time in both the kinematics and ecology units. Since the kinematics models had single agents, the challenges related to agent-aggregate relationships did not occur in Activities 1-4. Unlike the other three categories of challenges, the number of challenges did not increase in Activity 4, possibly because Activity 4 did not introduce any new agent-based thinking related challenges. However, the agent based thinking challenges resurfaced in Activity 5 when students were required to model multiple new agents, and modeling multiple agents caused the number of challenges to increase sharply. Like other types of challenges, students were also able to overcome most of these challenges by Activities 6 and 7.

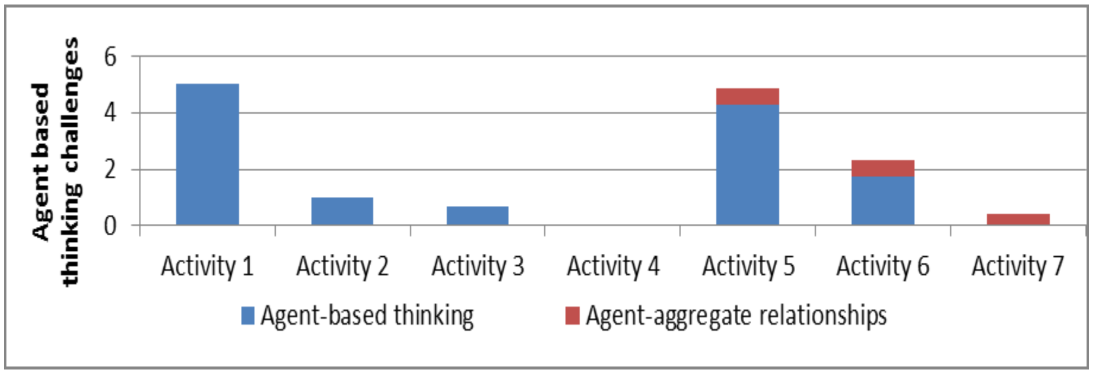


Figure 12. Average number and type of agent-based thinking challenges over time

4.4 Critical summary and implications for modifications to the CTSiM environment

In summary, this research study played an important role in evaluating the initial version of CTSiM, and documenting and analyzing the challenges faced by students when integrating CT with middle school science curricula using CTSiM v1. We showed that the intervention produced significant science learning gains in kinematics and ecology domains, especially when students were individually scaffolded by members of our research team based on their challenges in the CTSiM v1 environment. These gains could be considered a combined result of a number of factors like the CTSiM v1 system design, the activity progression from more simple, single-agent modeling activities to more complex, multi-agent modeling activities, and the one-on-one scaffolds provided to students whenever they faced difficulties. Our analyses show that the scaffolds generally helped reduce the challenges faced by students as they worked through a progression of activities in one domain, though some challenges resurfaced after initial scaffolding, primarily in activities where the number of complexities increased in comparison to previous activities.

Our results also contribute to the literature on CT at the K-12 level, where little is known about students' difficulties and developmental processes as they work in CT-based environments, especially CT-based environments that promote synergistic learning. Our results show that any learning-by-design CT-based environment needs to build in support for programming, domain knowledge acquisition, and modeling tasks. Also, challenges are not mutually exclusive, and taking this account may lead to developing more effective scaffolds. Programming and modeling challenges can be compounded by domain knowledge related challenges and can resurface in the context of new domain content. Therefore, scaffolds also need to focus on contextualizing programming and modeling scaffolds in terms of domain content.

In spite of being an initial usability study with a small sample size, this study served as an important first step towards making decisions on directions for redesign and further development of CTSiM v1. While the challenges identified may not be a comprehensive list and could possibly be categorized differently, the specific challenges and scaffolds we identified in this study played a vital role in laying the groundwork for redesigning the CTSiM v1 environment and integrating adaptive scaffolding to help students simultaneously develop a strong understanding of both CT and science concepts. We have substantially modified the CTSiM v1 interfaces and added new tools and features to help alleviate some of the students' challenges that we identified in this study.

For example, to help students deal with their modeling challenges related to representing a science domain in the multi-agent based modeling paradigm (MABM) and identifying the entities in the science domain and their interactions, we have modified the model construction process in CTSiM by making students model at two levels of abstraction, starting with a more abstract conceptual model of the domain. We have developed new interfaces to help students conceptualize science phenomena in the MABM paradigm, before they start constructing their computational models in the Build interface. The existing 'Build' interface has also been modified requiring students to conceptualize each agent behavior as a sense-act process (properties that are sensed and properties that are acted on) before building the block based computational model for the behavior. We have added dynamic linking between these representations for conceptual and computational modeling, emphasizing important CT practices of decomposing a task, modeling at different levels of abstractions and understanding relations between abstractions.

Also, to help students overcome their domain knowledge challenges, we have developed hypertext science resources for the kinematics and ecology units, and made them available in the CTSiM environment. Similarly, to help students with understanding programming constructs, flow of control, and the agent based modeling paradigm, we have made available a second set of hypertext resources, which we call the 'Programming guide'. In addition, we have added tools to help students check their understanding of important science and CT concepts through multiple-choice based formative quizzes.

Further, we have been working on adding scaffolding tools to support students in their model validation and debugging tasks. For instance, we have added model tracing capabilities so that students can step through each programming construct in their models and observe the individual effects of each of the constructs on the student model generated simulations. Similarly, we have added a code commenting feature, where students can opt to uncheck or comment out certain sections of their model to observe effects of individual programming constructs or code snippets. Also, we now provide students with the opportunity to compare their model in parts by comparing one or more agent behaviors at a time.

Finally, besides making substantial modifications to the CTSiM environment by adding new interfaces and tools, we have designed adaptive scaffolding that accounts for how students use the different tools and combine information from the different interfaces, and helps students, based on their observed deficiencies, in using good modeling strategies and building correct science models. We have conducted research studies with this newer version of CTSiM used in classroom settings, and found that the modified environment along with the adaptive scaffolds result in higher learning gains than those in our initial study where students received individualized one-on-one verbal scaffolds from human researchers. We have also compared students' behaviors and performances with and without the adaptive scaffolding, and demonstrated the effectiveness of the adaptive scaffolding framework in terms of students' science and CT learning, and their modeling performances and behaviors. Chapters 5 and 6 describe the modifications made to the initial CTSiM environment based on challenges observed in this initial interview-based study, and Chapter 7 reports the research study conducted to evaluate the effectiveness of the newly designed adaptive scaffolding framework.

CHAPTER 5

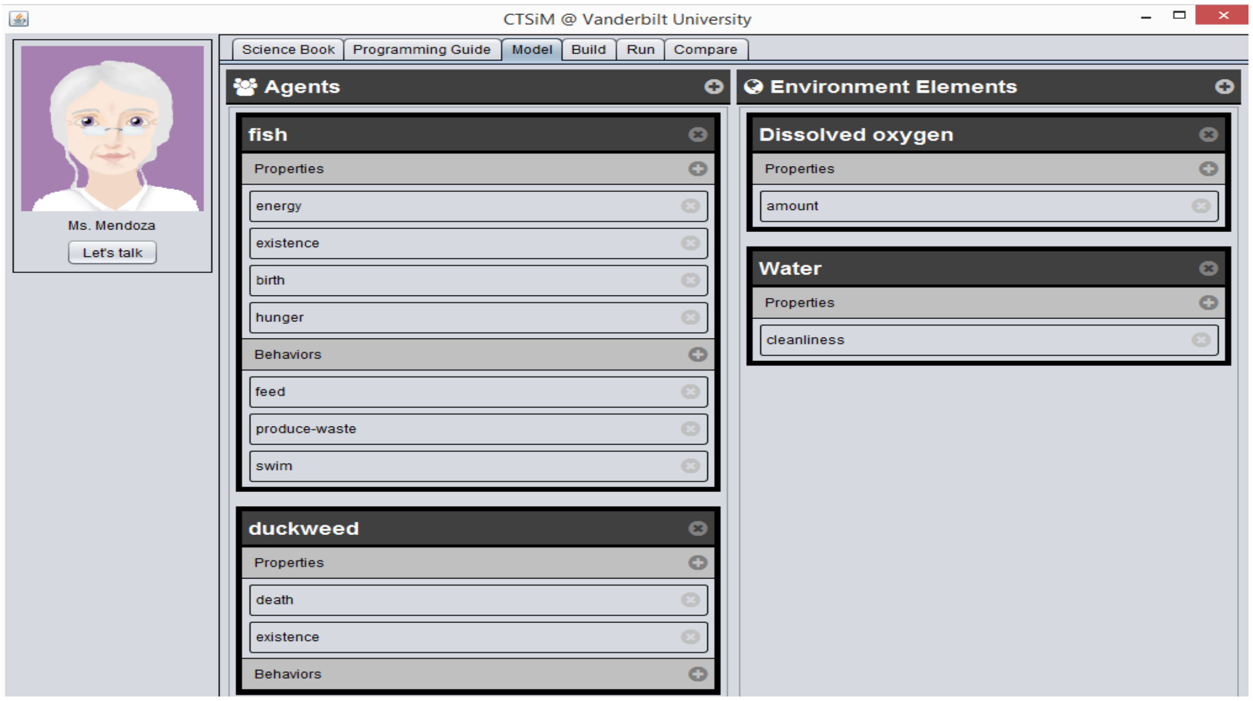
Modifications to the CTSiM environment: System-based scaffolds

In order to help alleviate some of students' difficulties identified in the previous chapter, we made a number of design revisions to the existing CTSiM environment. This included implementing new interfaces, modifying old ones, and adding a pedagogical mentor agent and new tools and features to scaffold different aspects of students' learning-by-modeling processes, i.e., information acquisition, model construction, and model verification. In addition, logging functionalities were added so that more details of students' actions with the different tools provided in the CTSiM environment, such as temporal information and context in which the actions were performed could be captured for post hoc analyses. We will, henceforth, refer to this new version of the system as CTSiM v2. We developed CTSiM v2 by iterative refinement of CTSiM v1, influenced by observations and analyses from research studies using CTSiM v1 and intermediate versions of the CTSiM environment.

5.1 Multiple linked representations for model building in CTSiM

We modified the model building task in CTSiM to help students overcome their modeling challenges identified in the previous chapter. These challenges related to identifying relevant entities, their properties, and their behaviors, and choosing the correct preconditions to model agent behaviors and interactions among the entities in the system. To deter students from using a trial-and-error approach to modeling the agent behaviors using programming blocks, we extended the model building task to include two linked representations (Basu, Biswas & Kinnebrew, 2016). Students start with an abstract conceptual representation of the domain, where students explicitly identify the entities that make up the domain, and define their behaviors and interactions. As a second step, students leverage the conceptual model structures to construct block-based computational models that represent individual agent behaviors. Though there exists an implied hierarchical structure between the two representations, we allow students to switch between the representations so that they can construct and refine their models in parts.

It has been hypothesized that multiple external representations (MERs) facilitate the development of a deeper understanding of science phenomena, something that is harder to achieve with single representations (Ainsworth, 2006). The ability to construct and switch between multiple perspectives in a domain helps learners build abstractions that are fundamental to successful learning in the domain (Ainsworth & van Labeke, 2004). Furthermore, insights achieved through the use of MERs increases the likelihood of transfer to new situations (Bransford & Schwartz, 1999). However, studies on the benefits of MERs have produced mixed results, possibly due to the cognitive load that is imposed on novices when they work with MERs (Mayer & Moreno, 2002; Ainsworth, 2006). Learners have to understand the constructs and semantics associated with each representation, while also discovering the relations between these representations. Studies have shown that learners tend to treat representations in isolation and find it difficult to relate, translate between, and integrate information from MERs, (van der Meij & de Jong, 2006). To derive benefits from MERs, learners need additional support. Some common forms of support include an integrated presentation of the MERs that includes dynamic linking or translation between them (Ainsworth, 2006; Goldman, 2003). In CTSiM v2, we provide support for integrating and maintaining correspondence between the conceptual and computational modeling representations in a number of ways as explained below.



fish
- Properties
  - energy
  - existence
  - birth
  - hunger
- Behaviors
  - feed
  - produce-waste
  - swim

duckweed
- Properties
  - death
  - existence
- Behaviors

Dissolved oxygen
- Properties
  - amount

Water
- Properties
  - cleanliness

Figure 13. The conceptual modeling representation in CTSiM v2 for structuring the domain in terms of entities, their properties, and behaviors

In the CTSiM v2 conceptual model representation, shown in part in Figure 13, students model the science domain using a visual editor to identify the primary entities (agents and environmental elements) in the domain of study, along with the relevant properties associated with these entities. Students also identify agent behaviors and represent the behaviors in an abstract form as sense-act models by specifying the agent and environment properties that trigger the agent behavior, and the agent and environment properties that are affected as a result of the behavior. Representing agent behaviors as sense-act models helps identify the different interactions between agents, as well as between agents and environment elements. In the Ecology activity involving modeling a fish tank, 'fish' represents an agent with relevant properties like 'hunger' and 'energy' and behaviors like 'feed' and 'swim'; and 'water' represents an environment element with properties like 'amount' and 'cleanliness'. The 'fish-feed' behavior can be defined as: sense the properties 'fish-hunger' and 'duckweed-existence', and act on properties, such as 'fish-energy' and 'duckweed-death.' However, this representation abstracts several details of agent behaviors, e.g., how and when the different properties are acted on. These details are captured in the agent behaviors modeled using the block-based visual computational language we have developed for CTSiM.

Students construct their computational models in CTSiM v2, using the same block structures and drag and drop interface as in CTSiM v1. Also, like in CTSiM v1, the palette of programming blocks includes domain-specific and domain-general constructs. However, a primary difference in CTSiM v2 is that the properties specified in the sense-act conceptual model representation for an agent behavior determine the set of domain-specific primitives available in the programming palette for constructing the computational model for that behavior. This explicit link helps students take a top-down approach to building behavior models, which in turn, helps them gain a deeper understanding of the representations and their relationships. For example, the 'wander' block is available in the 'fish-swim' behavior, only if 'fish-location' is specified as an acted on property for the behavior. CTSiM adopts a single internal representation for specifying the agent-based conceptual and computational modeling constructs, and a sense-act framework that help students focus on concepts associated with a specific science topic, while also accommodating CT constructs that apply across multiple science domains.

Figures 13 and 14 depict the modeling representations. Figure 13 represents a part of the conceptual modeling interface known as the '*Model*' interface, where students structure the science topic in terms of its entities, and their properties and behaviors. Figure 14 represents a combined conceptual-computational interface, known as the '*Build*' interface for modeling agent behaviors ('fish-feed' in this case). The leftmost panel depicts the sense-act conceptual representation, while the middle panel shows the computational palette, and the right panel contains the student-generated computational model. The side-by-side placement of the representations is a conscious design decision to provide integrated presentation support and is based on the fact that learners find it easy to understand physically-integrated material, rather than separately presented material (Chandler & Sweller, 1992).

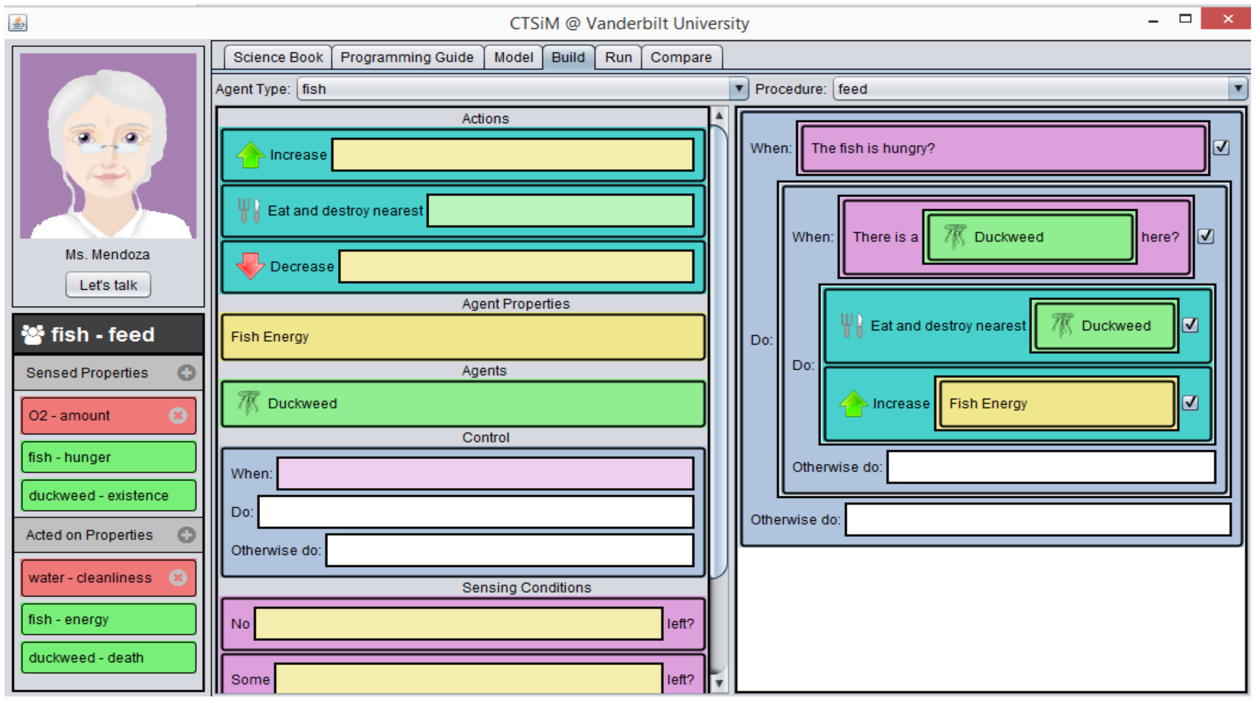


Figure 14. The linked conceptual-computational interface for modeling agent behaviors in CTSiM v2

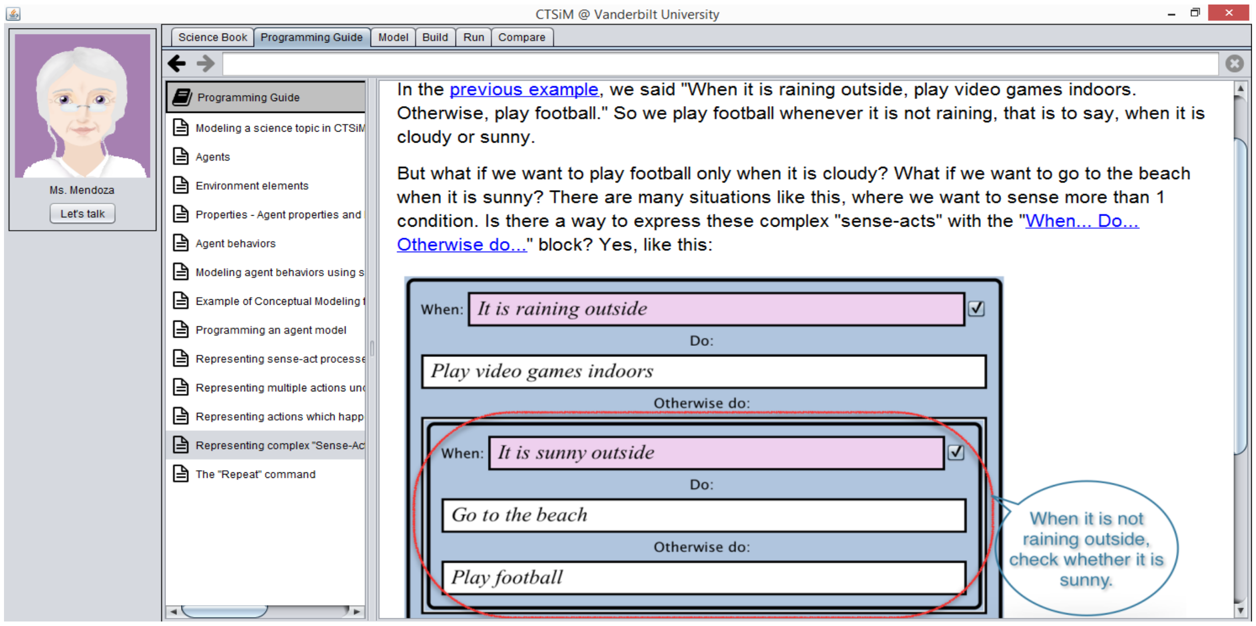
To further aid the integration, the red/green coloring of the sense-act properties (see Figure 14) provides students with visual feedback about how closely their computational models for an agent behavior correspond to their conceptual model for that behavior. Initially, the sense-act properties are colored red. As students build their computational model and add sensing and action blocks that correspond to these properties, they change color from red to green (another example of support provided by the dynamic-linking). For example, in Figure 14, the student has conceptualized that O_2-amount needs to be sensed for the fish-feed behavior. However, the computational model does not include this information and hence, the property is colored red. In such cases, students can verify individual agent behaviors and decide how to refine their computational and/or conceptual models.

Besides emphasizing conceptual understanding of science topics before modeling them computationally, this interface redesign operationalizes the important CT concepts of decomposition, abstraction, and understanding relations between abstractions. It provides a system-based scaffold for helping students understand two modeling representation at different levels of abstraction, decompose their modeling task between the two representations, and maintain correspondence between the two representations.

5.2 Information acquisition tools

In addition to providing system-based scaffolds for the model construction process, the CTSiM v2 environment also contains tools for acquiring information needed for successful model construction. In past work, we observed that students faced challenges (see Section 4.3.2) in understanding science content (domain knowledge challenges) as well as the semantics and use of computational constructs (programming challenges). To facilitate and support student learning, we developed separate resources for providing students with the relevant science and programming information. CTSiM v2 includes two sets of searchable hypertext resources, a '*Science Book*' with information about the science topic being modeled, and a '*Programming Guide*' with information about agent-based conceptual and computational modeling and use of CT programming constructs. While the contents of the '*Programming Guide*' remain constant across different domains and learning activities, the contents of the '*Science Book*' vary in accordance with the science topic being modeled. Figure 15 illustrates a page from the '*Programming Guide*' resources, while Figure 16 shows a sample page from the '*Science Book*' for the Ecology unit.

Currently, both sets of resources contain age-appropriate text with embedded hyperlinks and images. Some of the resource materials also include simulation code or animations that students can run. Both sets of resources also include a search functionality where students can search for pages with information about a particular term or phrase. Students can navigate to resource pages by following the table of contents, the hyperlinks on pages, or the search results. Details of the CTSiM v2 science and CT resources are contained in Appendix A. Students need to combine and apply information from both these resources to successfully build their agent-based computational models of science topics.


CTSiM @ Vanderbilt University
Science Book
Programming Guide
Model
Build
Run
Compare
Ms. Mendoza
Let's talk
Programming Guide
Modeling a science topic in CTSiM
Agents
Environment elements
Properties - Agent properties and
Agent behaviors
Modeling agent behaviors using s
Example of Conceptual Modeling f
Programming an agent model
Representing sense-act processe
Representing multiple actions und
Representing actions which happ
Representing complex "Sense-Ac
The "Repeat" command

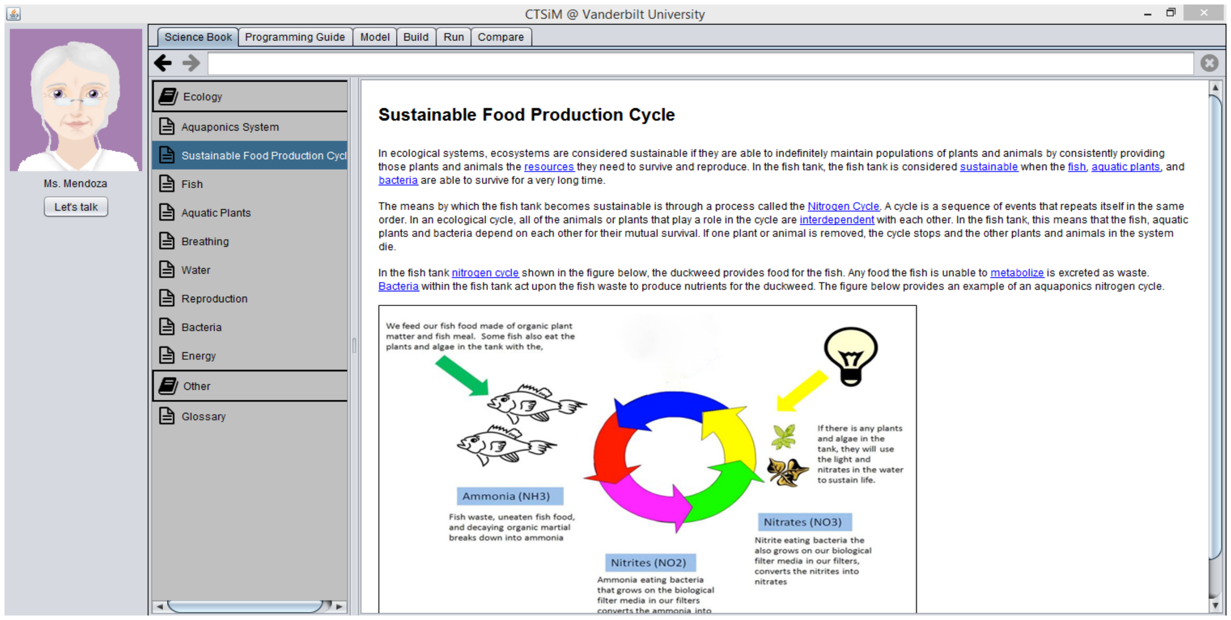
In the previous example, we said "When it is raining outside, play video games indoors. Otherwise, play football." So we play football whenever it is not raining, that is to say, when it is cloudy or sunny.

But what if we want to play football only when it is cloudy? What if we want to go to the beach when it is sunny? There are many situations like this, where we want to sense more than 1 condition. Is there a way to express these complex "sense-acts" with the "When... Do... Otherwise do..." block? Yes, like this:

```
When: It is raining outside
    Do:
    Play video games indoors
    Otherwise do:
        When: It is sunny outside
            Do:
            Go to the beach
            Otherwise do:
            Play football
```

When it is not raining outside, check whether it is sunny.

Figure 15. A screenshot from the CTSiM v2 'Programming Guide'



Sustainable Food Production Cycle

In ecological systems, ecosystems are considered sustainable if they are able to indefinitely maintain populations of plants and animals by consistently providing those plants and animals the resources they need to survive and reproduce. In the fish tank, the fish tank is considered sustainable when the fish, aquatic plants, and bacteria are able to survive for a very long time.

The means by which the fish tank becomes sustainable is through a process called the Nitrogen Cycle. A cycle is a sequence of events that repeats itself in the same order. In an ecological cycle, all of the animals or plants that play a role in the cycle are interdependent with each other. In the fish tank, this means that the fish, aquatic plants and bacteria depend on each other for their mutual survival. If one plant or animal is removed, the cycle stops and the other plants and animals in the system die.

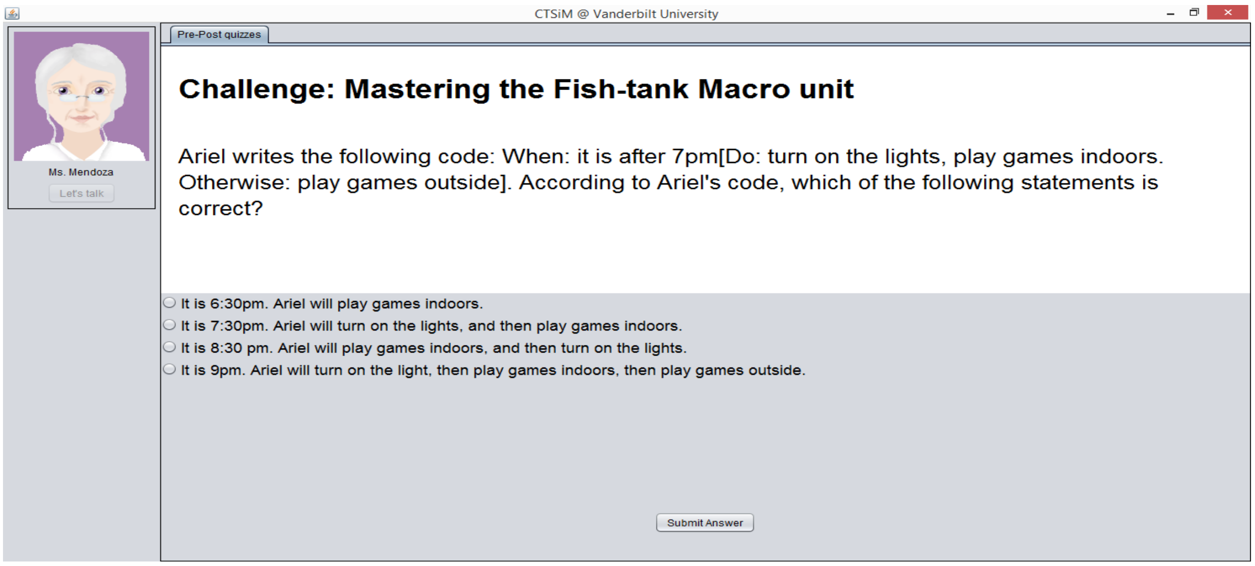
In the fish tank nitrogen cycle shown in the figure below, the duckweed provides food for the fish. Any food the fish is unable to metabolize is excreted as waste. Bacteria within the fish tank act upon the fish waste to produce nutrients for the duckweed. The figure below provides an example of an aquaponics nitrogen cycle.

We feed our fish food made of organic plant matter and fish meal. Some fish also eat the plants and algae in the tank with the,
If there is any plants and algae in the tank, they will use the light and nitrates in the water to sustain life.
Ammonia (NH3)
Fish waste, uneaten fish food, and decaying organic martial breaks down into ammonia
Nitrites (NO2)
Ammonia eating bacteria that grows on the biological filter media in our filters converts the ammonia into
Nitrates (NO3)
Nitrite eating bacteria the also grows on our biological filter media in our filters, converts the nitrites into nitrates

Figure 16. A screenshot from the CTSiM v2 'Science Book' for the Ecology unit

CTSiM v2 also offers students opportunities for checking their science and CT understanding through formative quizzes administered by a mentor agent introduced in the system named Ms. Mendoza. This mentor agent is positioned on the leftmost panel of the CTSiM v2 environment (see Figures 12-15), and students can opt to seek help and talk to her at any point during the modeling activity by clicking on the 'Let's talk' button below the mentor agent avatar. During a conversation between a student and Ms. Mendoza, all other CTSiM functionalities are temporarily made unavailable to the student till the conversation is over.

If a student decides to take a new quiz during a learning activity, Ms. Mendoza chooses a set of multiple-choice questions based on science and CT knowledge needed for the activity the student is working on, and grades students' responses. Figure 17 illustrates an example formative quiz question administered by Ms. Mendoza. If the student makes a mistake on a question, she points out the relevant page from the science or CT resources that needs to be read. Students can choose to take none or multiple formative quizzes in each unit. They can also review their last quiz taken and agent feedback received, or retake the questions they got wrong on their last quiz to improve their score.



Challenge: Mastering the Fish-tank Macro unit

Ariel writes the following code: When: it is after 7pm[Do: turn on the lights, play games indoors. Otherwise: play games outside]. According to Ariel's code, which of the following statements is correct?

- It is 6:30pm. Ariel will play games indoors.
- It is 7:30pm. Ariel will turn on the lights, and then play games indoors.
- It is 8:30 pm. Ariel will play games indoors, and then turn on the lights.
- It is 9pm. Ariel will turn on the light, then play games indoors, then play games outside.

Figure 17. An example formative assessment quiz question administered by Ms. Mendoza in the CTSiM v2 environment

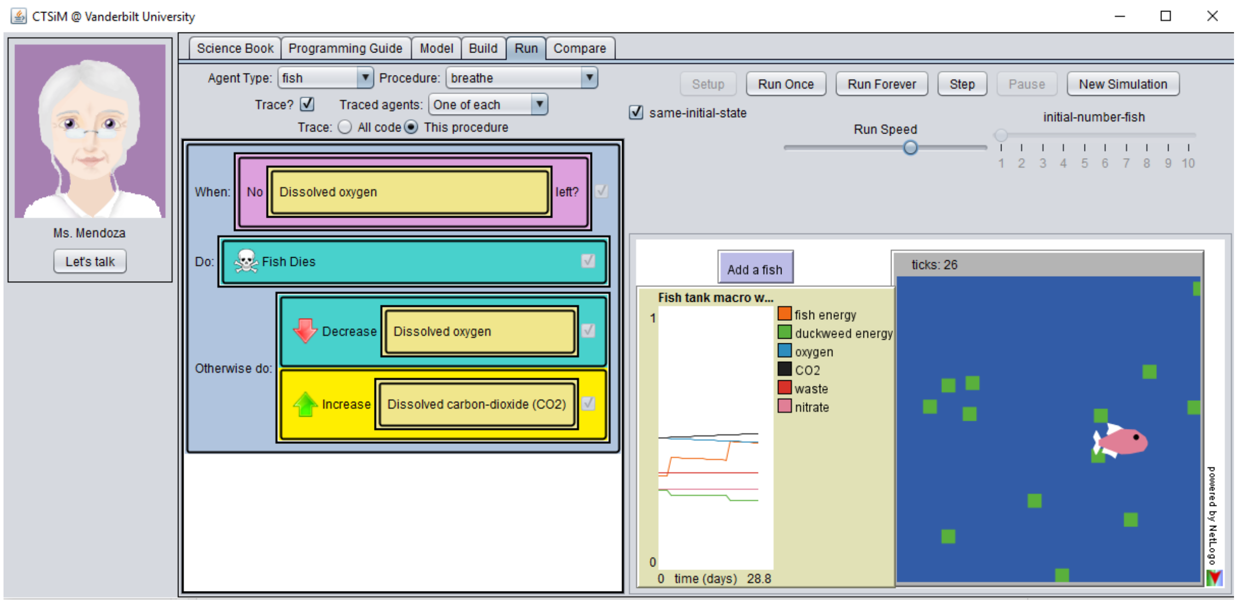
Based on students' responses during an activity, Ms. Mendoza maintains a record of students' proficiencies on the different set of science and CT concepts and knowledge components needed for the activity. Students' proficiencies in any science or CT concept are determined through their answers to questions testing the concept. Once a student demonstrates understanding of a concept by answering three questions related to the concept correctly, Ms. Mendoza infers that the student understands that concept, and moves the student to another concept. If a student demonstrates proficiency on the entire set of science and CT concepts linked to an activity, Ms. Mendoza informs the student that s/he has mastered the necessary concepts and needs to focus on applying them to build the correct model.

5.3 Model debugging tools

In order to help students with a number of their challenges described in Section 4.3.2 (Basu et. al., 2013) related to modularizing and debugging agent behaviors (programming challenges), specifying model parameters, understanding model component behaviors, systematically verifying models (modeling challenges), and understanding agent-aggregate relationships (agent-based-thinking challenges), CTSiM v2 provides students with a number of tools in the '*Build*', '*Run*', and '*Compare*' interfaces. All the tools are designed to help students debug their models in parts, understand the effects of individual agents, agent behaviors, and individual programming constructs within an agent behavior.

When students build their computational models in the '*Build*' interface, each block they drag from the programming palette onto the model building pane is appended with a checkbox containing a tick mark (see Figure 14). Programming blocks with a ticked checkbox are included in the student model to be executed. CTSiM v2 offers students the opportunity to comment out one or more programming blocks by unchecking the boxes, thus supporting execution of subsets of the student model. This commenting feature helps students easily modify their models and better understand the effects of particular programming blocks or code snippets, without having to physically add and remove blocks.

In addition, the CTSiM v2 '*Run*' interface supports model tracing, meaning that the system can highlight each primitive in the student's model, as it is executed and visualized as an agent-based simulation. Figure 18 illustrates the current 'Run' interface with an option for turning the 'Trace' functionality on or off. Once the 'Trace' checkbox is ticked and set to true, students are allowed to choose whether they want to trace all their code or a particular procedure, and whether they want to trace all agents, or one of each modeled agent-type, or a particular watched agent. For example, in Figure 18, the tracing option has been set to trace the 'fish-breathe' procedure and one of each agent-type. In order to achieve normal speed of execution during model tracing, each visual primitive is translated separately to NetLogo code via a model interpreter, instead of the entire user model being translated to NetLogo code. Such system-scaffolds for making algorithms "live" are directed at helping students better understand the correspondence between their models and simulations, and the effects of specific model components and parameter values, in turn supporting identification and correction of model errors.


Agent Type: fish
Procedure: breathe
Trace? ✓
Traced agents: One of each
Trace: ○ All code [● This procedure]
✓ same-initial-state

When: No Dissolved oxygen left?
Do: Fish Dies
Otherwise do: Decrease Dissolved oxygen
Increase Dissolved carbon-dioxide (CO2)

Figure 18. The CTSiM v2 'Run' interface with model tracing functionality

Furthermore, we updated the '*Compare*' interface where students verify their science models by comparing their model simulations against provided expert simulations. CTSiM v2 allows students to compare one or more agent behaviors at a time against the corresponding set of expert behavior(s), instead of having to always compare their entire model against the entire expert model. We believe this feature will be particularly useful in complex units with multiple agents and multiple agent behaviors where testing in parts can prove to be a useful model verification and debugging strategy. Figure 19 illustrates the modified '*Compare*' interface in CTSiM v2 with a checkbox allowing students to compare a subset of agent behaviors. The checkbox is not ticked (set to false) by default, meaning the entire student model will be compared against the entire expert model. If a student chooses this option, they are provided with a list of agent behaviors they have specified in the *'Model'* interface, and are required to choose one or more agent behaviors to compare from this list.

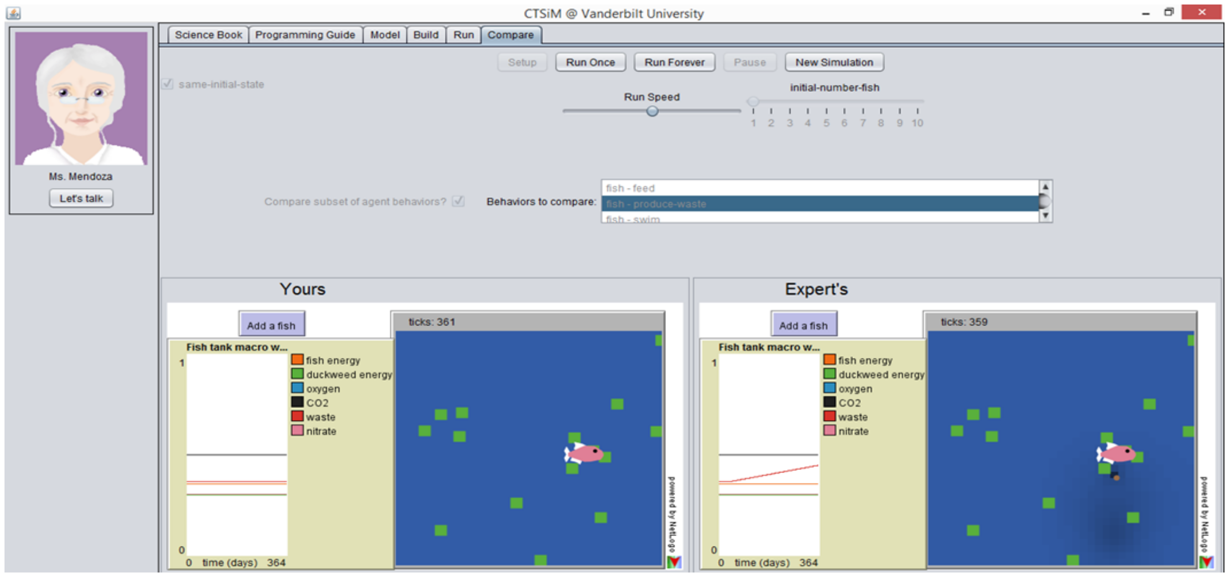


Figure 19. The CTSiM v2 'Compare' interface with functionality for verifying the model in parts

5.4 Summary

In summary, extensive design modifications were made to the initial CTSiM v1 interface based on students' observed challenges (Basu et. al., 2013; Basu et. al., in review), which resulted in the development of CTSiM v2. The modified learning environment scaffolds students' model-building processes using desired CT practices, provides students with the information required for building agent-based conceptual and computational models of science topics, and supports students with their model verification and debugging tasks. Additionally, CTSiM v2 added functionality for more detailed logging of student actions. As students use the different interfaces and tools available to them and interact with the mentor agent, all their actions are logged in sequence along with associated temporal information and other contextual information relevant to analyzing student behavior that can be inferred from the action sequences. For example, if a student drags and adds a block to her model, a 'Block added' action is logged and captures the system-time when the action occurred, the agent-procedure which was being modeled, the name and category of the block added, whether the block was added as a parent block or as a parameter for another block, and the resultant state of the student-model. This log data will help analyze students' learning behaviors and modeling processes and strategies, providing additional assessment measures beyond conventional pre- and post-tests. Details of post-hoc assessments using students' log data are provided in Chapter 7.

CHAPTER 6

Modifications to the CTSiM environment: Introducing adaptive scaffolds

While the CTSiM v2 design incorporates a number of scaffolds for information acquisition, model conceptualization and construction, and model assessment, novice learners may still find it difficult to use all of the tools and scaffolds in an effective manner to build and verify their models. The learning environment is open-ended, meaning that students can exercise their choices in the way they decompose, plan, sequence, and solve their given tasks.

Open-ended learning environments or OELEs (Clarebout & Elen 2008; Land et al., 2012; Land, 2000) are learner-centered computer environments designed to support thinking-intensive interactions with limited external direction. OELEs typically provide a learning context and a set of tools to help students explore, hypothesize, and build solutions to authentic and complex problems. The complex nature of the problems requires students to develop strategies for decomposing their problem solving tasks, apply them to develop and manage generated plans, and then monitor and evaluate the solutions that evolve from their plans. Thus, OELEs offer powerful learning opportunities for developing metacognitive and self-regulation strategies (Schwartz & Arena, 2013), all important components to prepare students for future learning (Bransford &Schwartz, 1999). However, the open-ended nature of these environments and the large number of choices for generating solutions available to novice learners produces significant challenges. Novice learners may lack proficiency in using system tools and the experience and understanding necessary to regulate their learning and problem solving in the environment. As a result, learners tend to adopt suboptimal learning strategies and make ineffective use of system tools, often failing in their open-ended learning tasks (Land, 2000). Hence, adaptive scaffolding is essential to help learners overcome these difficulties (Puntambekar and Hubscher, 2005).

In this chapter, we describe a task- and strategy-based scaffolding framework for interpreting and analyzing students' actions and activity sequences in OELEs, and how we use it to provide adaptive scaffolding in the CTSiM v2 environment. The goal of our adaptive scaffolds in CTSiM v2 is not to merely provide corrective feedback on the science models students build, but also to offer useful strategies for students' model building, model checking, and information acquisition behaviors. For example, when needed, CTSiM may scaffold students on model building strategies, such as seeking information relevant to the part of the model being built or tested, building and testing the model in parts, and modeling a topic conceptually to understand the scope of the model and the interactions between its components before trying to construct more detailed computational models. Going beyond several existing environments, our emphasis is on helping students gain insights into systematically building and testing their models, conducting meaningful analyses to discover the reason(s) behind their incorrect model behaviors, and applying systematic approaches to correcting their models. Systematic model building, debugging, and model correction skills are transferrable to a variety of scenarios and domains, thus making the learning generalizable.

6.1 Background review on adaptive scaffolding in OELEs for science or CT

Despite the well-recognized need for adaptive scaffolding in OELEs, a number of OELEs merely include non-adaptive support - tools and system-based scaffolds, like guiding questions, argumentation interfaces, workspaces for structuring tasks, data comparison tools, and tools for observing effects of plans made or models built. As Puntambekar and Hubscher (2005) point out, such tools are described as scaffolds, and they provide novel techniques to support student learning. However, such tools neglect important features of scaffolding, such as ongoing diagnosis, calibrated support, and fading. As a result, these tools are often unable to support the low performing novice learners who may be overwhelmed by the complexity of the task(s).

Even in OELEs with adaptive scaffolding, most do not provide scaffolds that target students' understanding of domain knowledge, cognitive processes, and metacognitive strategies in a unified framework. For example, MetaTutor (Azevedo, 2005) measures student behaviors in terms of a set of factors, such as the number of hypermedia pages learners have visited and the length of time spent on each page, to decide when to provide adaptive scaffolds in the form of suggestions (e.g., "*You should re-read the page about the components of the heart*"). In Ecolab (Luckin and du Boulay, 1999) - a modeling and simulation based OELE, the scaffolding agent intervenes whenever students specify an incorrect relationship in their models and provides a progression of five hints, each more specific than the previous one, with the final hint providing the answer. Similarly, in AgentSheets – one of the very few CT-based environments that includes adaptive scaffolds, students are supported by an automatic assessment of the computational artifacts they build (games or science simulations). The CT patterns present in students' artifacts are compared against desired CT patterns for the artifacts and represented in terms of what is known as the Computational Thinking Pattern (CTP) graph. Co-Lab (Duque et. al., 2012), on the other hand, tracks student actions to provide feedback about students' solutions (the models built by students) and work processes, but its feedback is limited to reminding students about specific actions they have not taken or should employ more frequently for model building and testing tasks. Action sequences or relations between actions are not analyzed and actions are not evaluated in terms of their consequences on the nature of the models constructed by the students.

Providing relevant adaptive scaffolding in OELEs requires interpreting learners' activities in terms of their cognitive skill proficiency and their use of metacognitive strategies for planning and monitoring. To facilitate adaptive scaffolding in OELEs and provide a framework that encompasses the cognitive and metacognitive processes associated with students' learning and problem solving tasks, we use a task- and strategy-based modeling framework combined with coherence analysis to interpret and analyze students' actions and action sequences in OELEs (Kinnebrew, Segedy, & Biswas, 2016 (in press); Segedy, Kinnebrew & Biswas, 2015). Our adaptive scaffolding framework is not based just on student performance (the accuracy of the models students build), and is not designed to provide students with bottom-out-hints if they fail to accomplish a task step (Koedinger & Aleven, 2007).

6.2 A task and strategy based adaptive scaffolding framework for OELEs

As discussed earlier, OELEs allow learners to exercise their choices in applying skills and strategies for decomposing their learning and problem solving tasks, and developing and managing their plans for accomplishing the learning tasks. The large solution spaces that can be attributed to the open-ended nature of such environments and the complexities of the search space clearly make the application of traditional overlay and perturbation modeling techniques (Sison & Shimura, 1998; Weber & Specht, 1997) intractable in such scenarios. The overlay approach to student modeling assumes that the student's knowledge is a strict subset of the expert knowledge included in the domain module, while the perturbation based modeling approach extends overlay modeling to account for bugs and misconceptions the student may have. Learner-based modeling approaches (Elsom-Cook, 1993) that focus more on learning behaviors and their impact on learning and evolution of the problem solution are likely to be more appropriate for OELEs. To facilitate learner-based modeling, and provide a framework that encompasses the cognitive and metacognitive processes associated with students' learning and problem solving tasks, we use a task- and strategy-based modeling framework to interpret and analyze students' actions and activity sequences in OELEs (Kinnebrew, Segedy, & Biswas, 2016 (in press);Segedy, Kinnebrew & Biswas, 2015).

At the core of this representational approach is a hierarchical task model, illustrated in the right half of Figure 20. The task model relates specific OELE activities to relevant tasks and ultimately to general tasks applicable across a variety of domains. Thus, the highest layers in this model include domain-general tasks that the learner has to be proficient in to succeed in a variety of OELE environments; and the middle layers linked to the higher layer, focus on approaches for successfully executing a set of subtasks, which may be specific to a particular OELE or genre of OELEs. Lower levels of the hierarchy map onto actions that are defined with respect to the tools and interfaces in a specific OELE. These actions are directly observable, and are typically captured in log files as students work on the system. Thus, the task model, which is represented as a directed acyclic graph, provides a successive, hierarchical breakdown of the tasks into their component subtasks and individual OELE actions. However, the task model does not indicate whether (or in what circumstances) multiple subtasks need to be completed to effectively perform a higher-level task, nor whether there are any necessary relations (such as an ordering) among them.

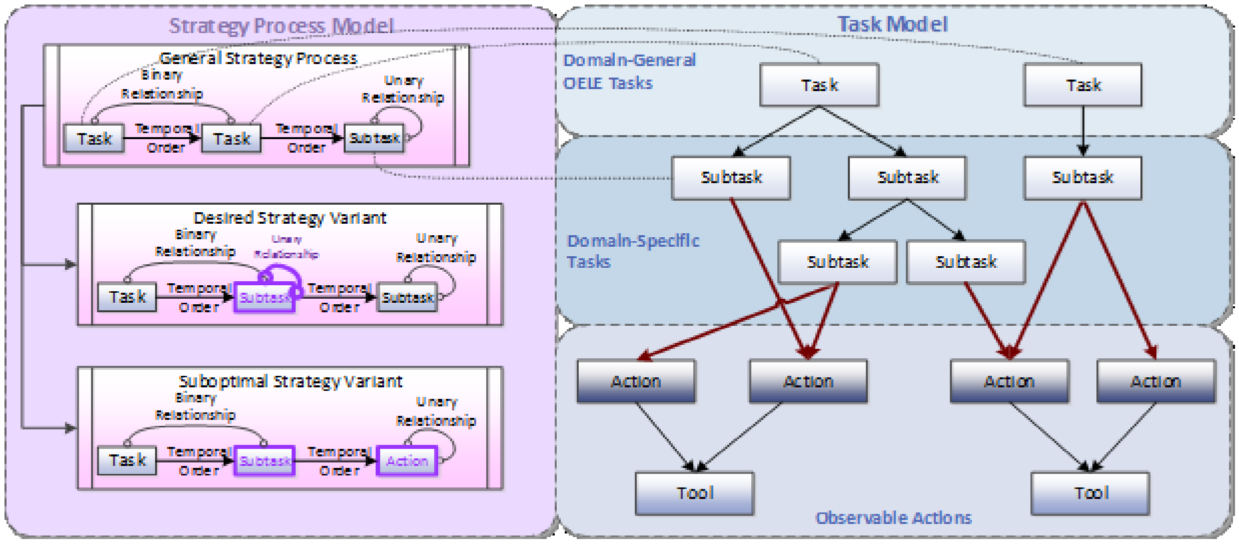


Figure 20. A task- and strategy-based modeling framework for OELEs

Instead, the strategy model, illustrated in the left half of Figure 20, captures information about action sequences and ordering between actions in a form that can be directly leveraged for online interpretation of a student's actions. The strategy model complements the task model by describing how actions, or higher-level tasks and subtasks, can be combined and/or associated with conceptual relationships to provide different approaches or strategies for accomplishing learning and problem-solving goals. By specifying a temporal order and conceptual relationships among elements of the task model that define a strategy, the strategy model codifies the semantics that provide the basis for interpreting a student's actions beyond the categorical information available in the task model.

Strategies have been defined as consciously-controllable processes for completing tasks (Pressley et al., 1989) and comprise a large portion of metacognitive knowledge; they consist of declarative, procedural, and conditional knowledge that describe the strategy, its purpose, and how and when to employ it (Schraw et al., 2006). How to perform a particular task in the OELE describes a *cognitive strategy*, while strategies for choosing and monitoring one's own cognitive operations describe *metacognitive strategies*. In this task-and-strategy modeling approach, strategies manifest as partially-ordered sets of elements from the task model with additional relationships among those elements determining whether a particular, observed learner behavior can be interpreted as matching the specified strategy. Figure 20 illustrates unary relationships that describe specific features or characterizations of a single strategy element, binary relationships among pairs of elements, and the temporal ordering among elements of the strategy. Further, if a relationship is not specified between any two elements in a strategy, then the strategy is agnostic to the existence or non-existence of that relationship. Because the elements of the task model used in the definition of strategies are hierarchically related, strategies may also naturally be related from more general strategy definitions to more specific variants. In this representation, specifying additional relationships, additional elements, or more specific elements (e.g., a specific action replacing a more general task/subtask) derive a more specific strategy from a general one, as illustrated in Figure 20.

An important implication of the hierarchical relationships among the strategy process definitions is that multiple variations on a more general process can automatically be related to each other. In particular, the hierarchical relations enable relating a set of desired and suboptimal implementations of a general strategy process for use in the learner model. As illustrated in Figure

20, the general outline of the strategy is hierarchically linked to a variety of more detailed versions of the process that represent either desired variants or suboptimal ones. By analyzing a student's behavior, the system can compare strategy matches to desired versus suboptimal variants in order to estimate the student's proficiency (and need for scaffolding) with respect to the strategy, as illustrated in Figure 21.

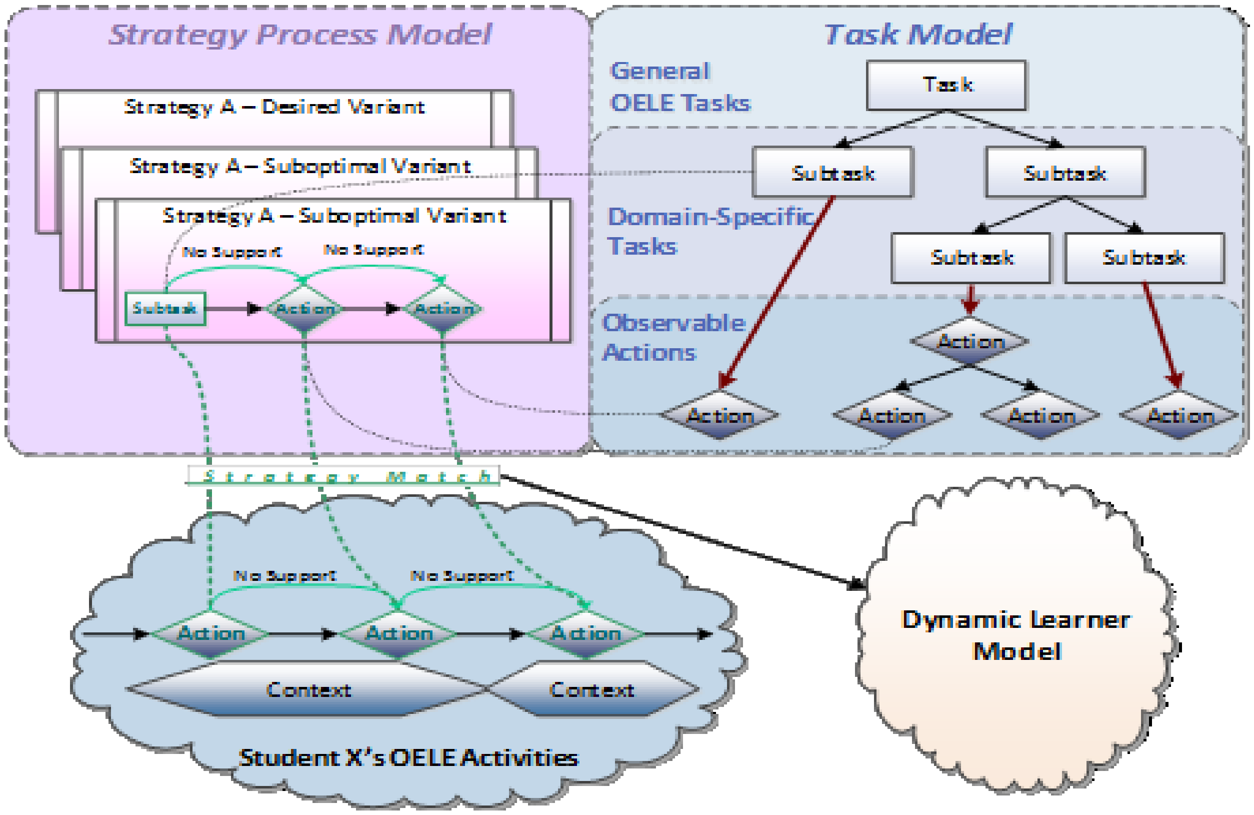


Figure 21. Strategy matching in OELEs

6.3 The CTSiM task and strategy models

We apply the generalized task- and strategy-based modeling framework described in Section 6.2 to interpret and analyze students' actions in the CTSiM v2 environment and accordingly provide them with adaptive scaffolds. The CTSiM task model hierarchy is shown in Figure 22. The top level of the model covers three broad classes of tasks that are relevant to a large number of OELEs: (i) *information seeking and acquisition*, (ii) *solution construction and refinement*, and (iii*) solution assessment*. Each of these OELE task categories is further broken down into three levels that represent: (i) general task and subtask descriptions that are common across the specific class of

OELEs that involve learning by modeling; (ii) CTSiM specific descriptors for these tasks; and (iii) actions within the CTSiM v2 environment that students use to execute the various tasks.

Information acquisition (IA) involves identifying relevant information, interpreting that information in the context of a current task or subtask (e.g., solution construction and refinement), and checking one's understanding of the information acquired in terms of the overall task of building correct models. In CTSiM v2, students are provided with separate searchable hypertext resources that contains the relevant science content information, and information and examples about conceptual and computational modeling and uses of CT constructs (see Section 5.2 for details). Students combine information from the two types of resources to build their science models using an agent based modeling approach, and use computational constructs to model agent behaviors using a sense-act framework. Students can check their understanding of the information acquired by taking quizzes provided in the system by the mentor agent, Ms. Mendoza, and can then use the quiz feedback to identify science and CT concepts they need to work on, and the relevant sources of information (specific resource pages) for learning about the concepts.

Solution construction (SC) tasks apply information gained through information seeking and solution assessment activities to construct and refine the required solution. In CTSiM v2, the solution refers to the science model that the student can build in parts using linked conceptual and computational representations. As described in Section 5.1, conceptual model construction involves structuring the domain in terms of agents, environment elements, their properties and behaviors, as well as representing agent behaviors as sense-act processes. The computational model construction, which is linked to the conceptual model, represents the simulation model that comprises agent behaviors created by selecting and arranging domain-specific and computational programming blocks.

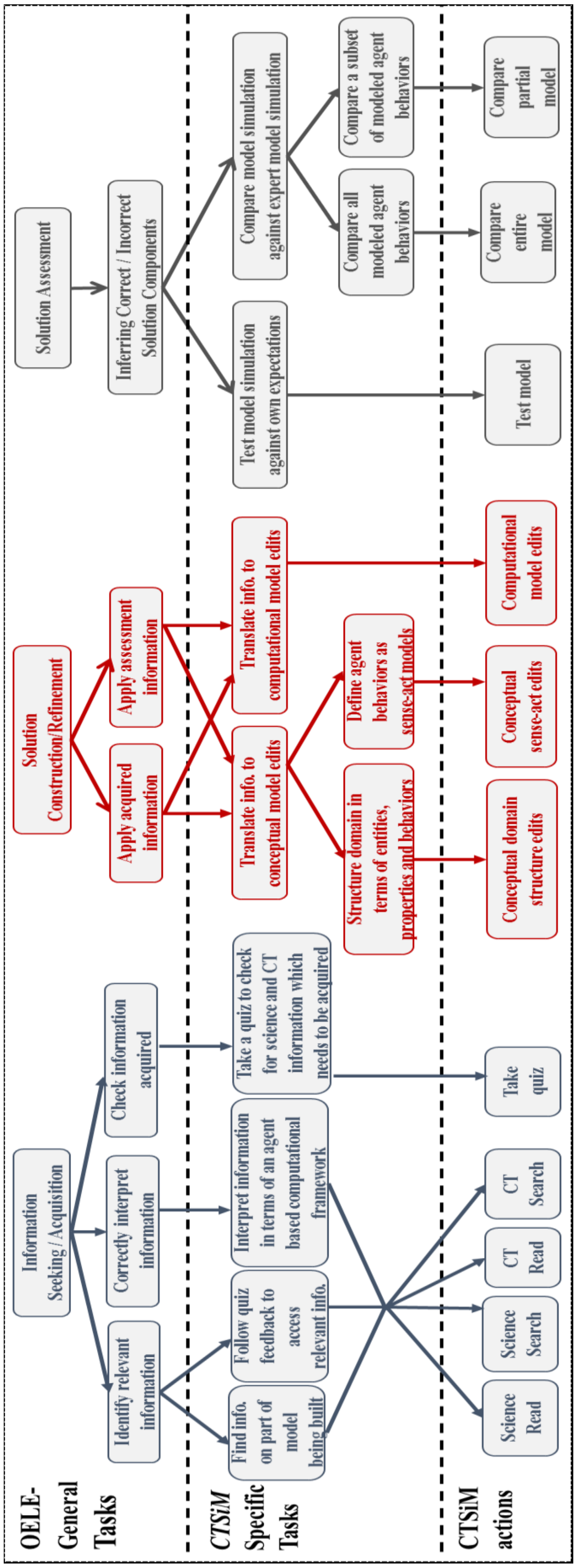

OELE-General Tasks
CTSiM Specific Tasks
CTSiM actions
Information Seeking / Acquisition
Identify relevant information
Correctly interpret information
Check information acquired
Find info. on part of model being built
Follow quiz feedback to access relevant info.
Interpret information in terms of an agent based computational framework
Take a quiz to check for science and CT information which needs to be acquired
Science Read
Science Search
CT Read
CT Search
Take quiz
Solution Construction/Refinement
Apply acquired information
Apply assessment information
Translate info. to conceptual model edits
Translate info. to computational model edits
Structure domain in terms of entities, properties and behaviors
Define agent behaviors as sense-act models
Conceptual domain structure edits
Conceptual sense-act edits
Computational model edits
Solution Assessment
Inferring Correct / Incorrect Solution Components
Test model simulation against own expectations
Compare model simulation against expert model simulation
Compare all modeled agent behaviors
Compare a subset of modeled agent behaviors
Test model
Compare entire model
Compare partial model

Figure 22. The CTSiM task model

Solution assessment (SA) tasks involve running simulation experiments, either in the 'Run' interface where students can step through their simulation code, and check the evolving model behavior by observing the animation and plots, or in the 'Compare' interface where students compare their model behaviors against an expert model behavior. The goal is to observe the behavior of the constructed model, and verify its correctness. SA tasks may involve testing the model in parts, comparing the results generated by the student's model against the behaviors generated by a corresponding expert model, and using this comparison to identify the correct and incorrect parts of the model. As discussed earlier, the student and the expert models are executed in lock step as NetLogo simulations. Observing and comparing the simulations helps infer incorrectly modeled agent behaviors, which students can combine with relevant information seeking actions to refine their existing solutions.

We use different sequences of these tasks, subtasks and actions described in the CTSiM task model, and combine them with information characterizing individual actions (unary relations) and relationships between different action sequences (binary relations) to specify a set of desired and suboptimal strategies or a 'strategy model' for CTSiM. While different unary relations can be used to characterize learners' cognitive processes, we use a unary measure called '*effectiveness*' (Segedy, Kinnebrew & Biswas, 2015) to evaluate learners' actions in the CTSiM v2 environment. Actions are considered effective if they move the learner closer to their corresponding task goal. For example, effective IA actions should result in an improvement in the learner's understanding of science and CT concepts required for model building in CTSiM. Likewise, effective SC actions improve the accuracy of learners' conceptual and computational models, and effective SA actions generate information about the correctness (and incorrectness) of individual agent behaviors modeled by the learner. Overall, students with higher proportions of effective actions are considered to have achieved higher mastery of the corresponding tasks and cognitive skills.

Similarly, many types of binary relations can be defined among tasks/actions to represent strategies. We adopt '*coherence*' metrics for defining effective strategies comprising action sequences (Segedy, Kinnebrew & Biswas, 2015). Two temporally ordered actions or tasks ($x \rightarrow y$), i.e., x before y, taken by a learner exhibit the coherence relationship ($x \Rightarrow y$) if x and y share contexts, i.e., the context for y contains information contained in the context for x. The context for an action comprises the specifics and details of the action, such as the specific science or CT page read for a '*Science Read*' or '*CT Read*' IA action, the particular conceptual or computational components edited and the part of the model worked on for SC actions, or the agent behaviors compared for SA actions. We can assume that students are more likely demonstrating effective metacognitive regulation when an action or task they perform is coherent with or relevant to information that was available in one of their previous actions or tasks.

In this version of CTSiM, we chose a set of five desired strategies (*S1*-*S5*), and analyzed students' actions to detect when students were not making optimal use of these strategies and needed scaffolding. Since there can be numerous ways in which a strategy may not be used optimally, we defined specific suboptimal variants of the strategies that we wanted to detect in students' learning behaviors, and provided feedback to help students overcome their deficiencies. For desired strategies associated with single actions, a suboptimal strategy use can involve an ineffective instance of the action or a lack of the action altogether. Similarly, for desired strategies involving coherent action sequences, suboptimal strategy use can be defined by the action sequence with component actions that are not coherent with each other, or by the lack of the action sequence itself.

We realize that the five strategies do not define a complete set of useful strategies for CTSiM, and several other strategies can be defined using actions and action sequences from the CTSiM task model. One way to define which strategies to detect and monitor could be using offline sequence mining techniques to analyze student behaviors, and then using frequent patterns of behavior derived from the offline analyses as strategies to detect in student behavior in future versions of the system (Kinnebrew, Loretz, & Biswas, 2013; Kinnebrew, Segedy, & Biswas, 2014). Our selection of strategies is based on our previous observations of challenges commonly faced by students (Basu et. al., 2013; Basu et. al., in review). Like we described in Section 4.3, we noticed that students needed repeated help with identifying the agents and their interactions in a science topic, understanding domain concepts and connecting them to the different CTSiM tasks, understanding how to represent science concepts using CT constructs, observing effects of partial code snippets, identifying differences between the user model simulations and the expert simulation, and debugging by decomposing the task into manageable pieces. While we have considerably modified the CTSiM interface based on our observations, we wanted to ensure we could provide addition individualized scaffolds when we detected that students were not using the tools and information sources in an efficient manner. In other words, information derived was not being used in an effective way for model building, combining conceptual and computational modeling, or debugging their model in parts.

Hence, three of the desired strategies, *S1*, *S2*, and *S3*, link SC and SA actions to IA actions. *S4* focuses on the complexities of SA for larger models, and describes a strategy for testing the model in parts. *S5* pertains to SC, and how to effectively use multiple linked representations to build the science model. Each of the desired strategies along with their suboptimal variants is discussed below.

Desired S1. Solution construction followed by relevant information acquisition strategy (SC-IA): This strategy relates to seeking information relevant to the part of the model currently being constructed by the student. It can be specified as a SC action (conceptual domain-structure edits, conceptual sense-act edits, or computational model edits) temporally followed by a coherent 'Science read' action (*SC ⇒ Science Read*). Coherence implies that the science resource page accessed contains information relevant to the agent or agent behavior being modeled in the SC action. For example, if a student switches to the science resources while modeling the sense-act structure of the 'fish-breathe' behavior, we can observe the desired (*SC ⇒ Science Read*) strategy only if the science resource pages read contain information about the 'fish-breathe' behavior.

Suboptimal S1: Suboptimal use of this strategy occurs when the part of the model the learner constructs has errors, and this is followed by the learner seeking information that does not correspond to the part of the model s/he was constructing. It can be specified as an *ineffective* SC action that is followed by a 'Science read' action *(ineffective SC → Science Read)*, which is *incoherent* with the previous SC action.

Desired *S2. Solution assessment followed by relevant information acquisition strategy (SA-IA)*: This strategy relates to seeking information relevant to the agent behaviors that were just assessed using a SA task (test model, compare entire model, or compare partial model). The IA that follows is required to be a coherent 'Science read' action (*SA ⇒ Science Read*), i.e., the science resource page contains information relevant to at least one of the agent behaviors assessed in the SA action.

Suboptimal *S2:* Suboptimal use of this strategy occurs when a SA action helps determine that one or more agent behaviors are incorrect, and the subsequent 'Science read' action *(effective SA detecting incorrect agent behaviors → Science Read)* is *incoherent*, i.e., it does not involve the reading of resource pages that are linked to the behaviors assessed to be incorrect.

Desired S3. Information acquisition prior to solution construction or assessment strategy (IA-SC/SA): This strategy involves acquiring information about an agent behavior before modeling it or checking that behavior of the agent. A 'Science Read' action that is followed by a coherent SC or SA action (*Science Read* $\Rightarrow$ *SC*|*SA*) defines this strategy.

Suboptimal S3: Suboptimal occurrence of this strategy occurs when a SA action finds an incorrect agent behavior, but this action is not temporally preceded by a 'Science Read' action for the incorrect agent behavior. The SA action can be temporally preceded by a 'Science Read' action for other agent behaviors (*incoherent* variant of the strategy), or it may not be preceded by any 'Science Read' action at all (lack of the action sequence).

Desired *S4. Test in parts strategy*: When a student's CTSiM model includes multiple agent behaviors, this strategy represents an approach where the student decides to assess a subset of the modeled behaviors at a time to make it easier to compare their model behaviors against the expert simulation. This strategy is characterized by the *effectiveness* of a single action, 'Compare partial model' in case of complex models where the expert model contains greater than 2 agent behaviors. An effective 'Compare partial model' action generates information about the correctness or incorrectness of individual or subsets of agent behaviors as opposed to the entire set of agent behaviors. We specify an effective 'Compare partial model' action as one that compares a maximum of 2 agent behaviors.

Suboptimal S4: A suboptimal use of this strategy occurs when a 'Compare' action during SA of a complex model with multiple erroneous agent behaviors does not provide sufficient information to find the source(s) of the errors. It can involve an *ineffective* 'Compare entire model' action or even an *ineffective* 'Compare partial model' action, which does not provide enough information to pin point errors to specific agent behaviors. An *ineffective* 'Compare' action involves a comparison of 3 or more agent behaviors for a student model with multiple incorrectly modeled agent behaviors.

Desired *S5. Coherence of Conceptual and Computational models strategy (Model-Build)*: This strategy involves maintaining the correspondence between the conceptual and computational models for each agent behavior. It can be represented as a 'Conceptual sense-act build' action followed by a coherent (linked) 'Computational model build' action (*Sense-act build* $\Rightarrow$ *Computational build*), i.e., the computational edit adds a sensing block corresponding to a sensed property or an action block corresponding to an acted property for the same agent behavior. As described in Section 5.1, we provide students with visual feedback about their conceptual-computational model coherence by coloring sense-act properties green or red based on whether the properties are coherently used or not used in their computational models. This visual information provides students feedback on how well they are employing the *Model-Build* strategy.

Suboptimal S5: Suboptimal uses of this strategy involve a conceptual sense-act edit action that is either not temporally followed by a computational edit action or is followed by an incoherent computational edit. An ineffective use of this strategy is detected through system based visual feedback about the sense-act properties for the agent behaviors. If the properties are colored red, it implies that this strategy was used in an ineffective manner.

6.4 Learner modeling for adaptive scaffolding in CTSiM

The primary goals of the adaptive scaffolding in CTSiM are to help students become proficient in the cognitive processes related to CTSiM tasks and subtasks, and the metacognitive processes represented as strategies that support efficient task execution. This form of feedback goes beyond the purely corrective and diagnostic approaches to feedback by shifting students' focus to monitoring their model building processes, leveraging the links between conceptual and computational models to systematically build complex models in parts, and developing the abilities to effectively test their evolving models by comparing against behaviors generated by a corresponding, but correct expert model.

To support this form of scaffolding, the CTSiM learner modeling framework is derived from the task and strategy based modeling framework discussed in Section 6.2. Our learner model represents a data-driven scheme that keeps track of students' performances on various tasks and related actions defined in the hierarchical task model, as well as the strategies they use to combine and co-ordinate the different tasks. Figure 23 illustrates a comprehensive learner modeling approach for CTSiM where the learner model maintains information about students' effectiveness on each of the IA, SC, and SA tasks, as well as their use of strategies that combine the IA, SC, and SA tasks in different meaningful ways. In the current CTSiM v2 system, we have designed detectors for and maintained information on a limited set of strategies, namely *S1-S5* as described in Section 6.3. Also, our current learner model primarily focuses on students' task performances for the SC tasks, i.e., conceptual and computational model building.

Learner actions in CTSiM, combined with information about the state of students' conceptual and computational models, are used to evaluate students' strategy use for *S1*-*S5*. For each of the five strategies, the 'strategy matcher' function in the 'learner modeling module' compares the corresponding suboptimal strategy variant against online learner information to calculate each learner's frequency of suboptimal strategy use. However, these frequency calculations are local, and count the frequency for suboptimal strategy use, since the last time the student was scaffolded on the particular strategy. The scaffolding module can directly use this information to decide when to provide strategy-based scaffolds. Our longer-term plan, however, is to store global information about students' strategy use, both for the desired and the suboptimal variants.

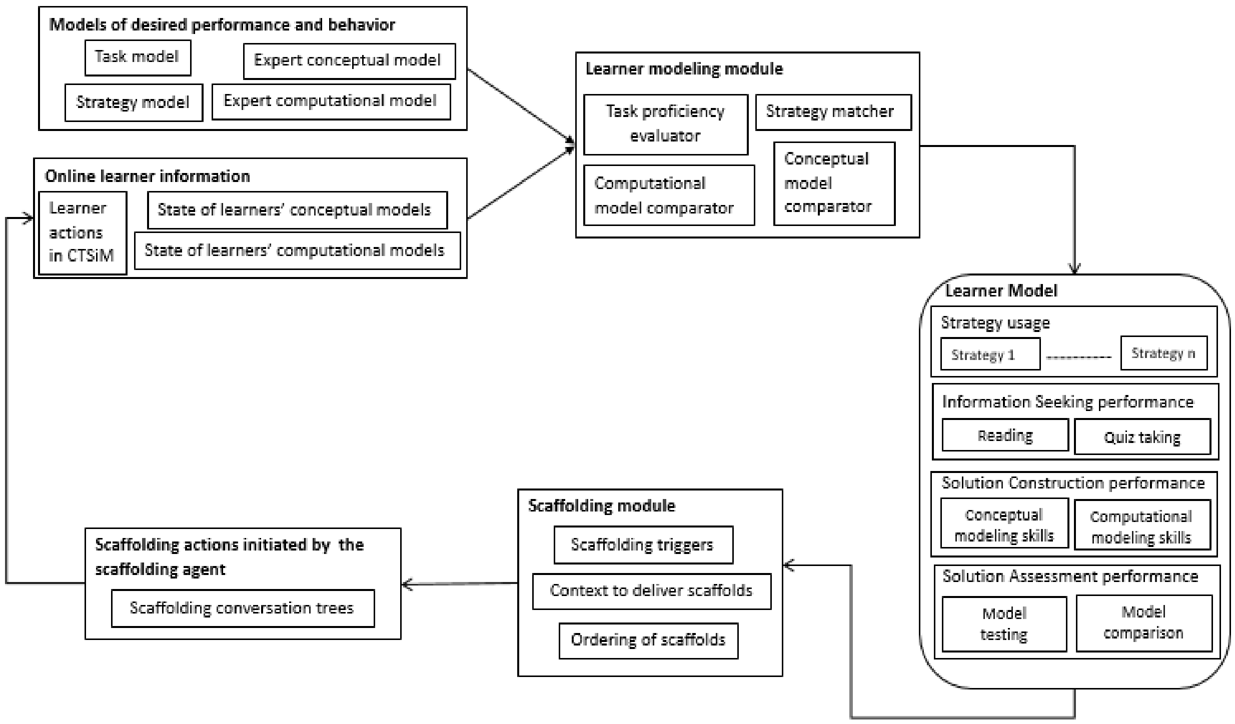

Models of desired performance and behavior
Task model
Expert conceptual model
Strategy model
Expert computational model
Online learner information
Learner actions in CTSiM
State of learners' conceptual models
State of learners' computational models
Learner modeling module
Task proficiency evaluator
Strategy matcher
Computational model comparator
Conceptual model comparator
Learner Model
Strategy usage
Strategy 1
Strategy n
Information Seeking performance
Reading
Quiz taking
Solution Construction performance
Conceptual modeling skills
Computational modeling skills
Solution Assessment performance
Model testing
Model comparison
Scaffolding module
Scaffolding triggers
Context to deliver scaffolds
Ordering of scaffolds
Scaffolding actions initiated by the scaffolding agent
Scaffolding conversation trees

Figure 23. Learner modeling and adaptive scaffolding framework for CTSiM

Besides maintaining a measure of learners' strategy use, the CTSiM learner model also maintains a local history of learners' SC task performances, i.e., conceptual and computational modeling skills. This helps detect ineffective SC actions and the aspects of the modeling tasks that learners are struggling with. Since ineffective SC edits can either remove model elements required in the expert model from the student model or add model elements not required in the expert model to the student's model, separate measures of 'missing/correctness' and 'extra/incorrectness' are maintained for students' conceptual and computational models and their various components. Learners' modeling skills are defined by measures comparing different aspects of their models against the corresponding expert models. Conceptual modeling skills are defined separately for different conceptual model components so that the scaffolds can focus on specific aspects of the modeling task. The different conceptual components include agents, environment elements, properties, and behaviors chosen, as well as the sensed and acted-on properties specified for each agent behavior. The conceptual model comparator in the learner modeling module performs a simple set comparison between students' conceptual models for a topic and the expert conceptual model for the topic to compute 'missing' and 'extra' measures for each of the conceptual model components, which are stored in the learner model. The 'missing' measure for a conceptual component counts the number of elements of the component that are present in the expert model, but are missing in the student model. Similarly, the 'extra' measure for a component counts the number of elements of the component that are present in the student model, but not present in the conceptual model.

On the other hand, defining computational modeling skills involve more nuanced measures beyond 'missing' and 'extra' blocks, since merely having the same set of programming blocks as the expert model does not guarantee semantic correctness of the student model. The same information can be modeled in a number of ways using different sets of blocks. While we cannot possibly account for all possible correct solutions, we have added a number of functions to our computational model comparator to minimize false positives (same set of blocks as expert model, but different semantic meaning) and false negatives (blocks do not match those in the expert model, but similar semantic meaning). For example, students can represent the correct information in different ways using different sets of blocks. If a conditional in a student model senses a property instead of its complement, or vice versa (e.g., using a 'some-left' block instead of a 'none-left' block), the consequent and alternative blocks can be interchanged to represent the same information. Another example of a false negative occurs when the expert model for an agent behavior contains a conditional and an action block that is independent of any condition and is hence placed outside the conditional block. If a student places two instances of the action block inside the conditional, once under the consequent and once under the alternative, the solution is less elegant, but conveys the same semantic meaning as the expert. The model comparator takes these possibilities into account, while determining ‘missing’ and ‘extra’ blocks. In order to account for false positives, the model comparator checks whether action blocks in the student models occur under the correct set of conditions as defined in the expert model (irrespective of any condition, under a particular sensing condition, or under multiple simultaneous sensing conditions). The comparator also checks properties whose values are increased or decreased in an expert agent behavior to make sure their direction of change is the same in the student model for the agent behavior. Otherwise, an expert model with blocks ‘Increase(CO_2-amount), Decrease(O_2-amount)’ will be equated to a student model with blocks ‘Increase(O_2-amount), Decrease(CO_2-amount)’, since both models have the same set of four blocks. In summary, the computational model comparator defines computational modeling skills for each agent behavior in terms of the following: (a) number of missing blocks in the behavior as compared to the expert model, (b) number of extra blocks in the behavior as compared to the expert model, (c) whether all actions in the behavior occur under the correct set of conditions (yes/no), and (d) whether all property values modified in the behavior were changed in the correct direction (yes/no).

While we can capture the state of students’ conceptual and computational models as they work in the CTSiM environment, we calculate and update the measures describing students’ conceptual and computational modeling skills in the learner model only when students assess their models. This design decision was made so that the scaffolds for the model building tasks were not sensitive to the effects of individual SC edits, but depended on the evolution of students’ models between model assessments. Also, since we have designed our scaffolds to depend on how students’ models evolve since the last model assessment, the learner model only maintains a local history of students’ modeling skills instead of maintaining a global one. In this version, the learner model stores a history of a student’s conceptual and computational modeling skills since the last time s/he was provided a scaffold for the particular model construction task.

6.5 Designing and Delivering adaptive scaffolds

Students’ task performance and strategy use information captured in the leaner model is used by the scaffolding module and combined with information about triggering conditions (frequency threshold for triggering particular scaffolds and priority and ordering of scaffolds) to decide which task based or strategy based scaffold to provide. Each scaffold is delivered in the form of a mixed-initiative conversational dialog initiated by the mentor agent - Ms. Mendoza, and is anchored in the context of students' modeling goals, their recent actions, and information that is available to students at that point of time (e.g., simulation information or domain information). The mixed-initiative, back-and-forth dialogues between the student and Ms. Mendoza are implemented as conversation trees. The root node of the tree represents Ms. Mendoza's initial dialogue, which then branches based on conversational choices available to the student. Ms. Mendoza can respond to students' choices using conversational prompts or by taking specific actions in the system. Such a structure captures the possible directions that a single conversation might take once it has been initiated. This conversation format engages students in a more authentic social interaction with Ms. Mendoza, and allows them to control the depth and direction of the conversation within the space of possible conversations provided by the dialogue and response choices (Segedy, Kinnebrew & Biswas, 2013). Figure 24 provides an example of a scaffolding conversation tree for the IA-SC/SA scaffold asking students to read about incorrectly modeled agent behaviors that they have modeled and assessed without reading. It illustrates how Ms. Mendoza and students can together negotiate goals and plans using such mixed-initiative conversational dialogues.

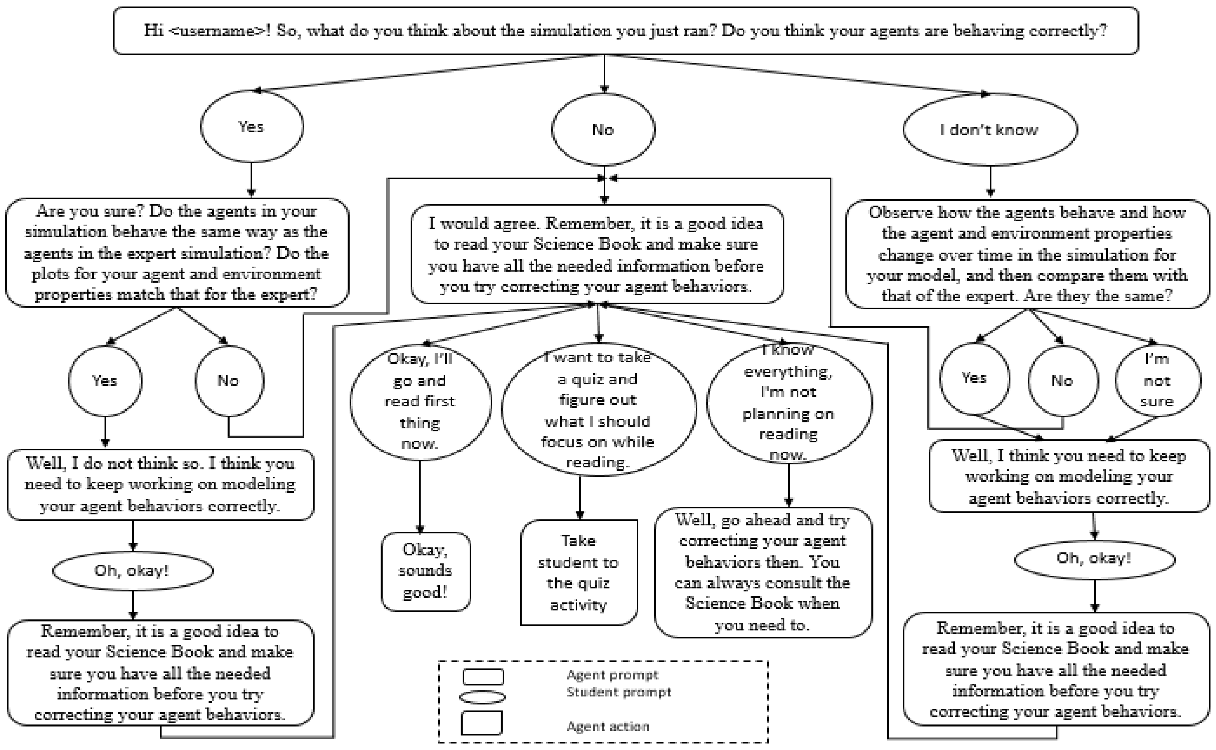

Hi <username>! So, what do you think about the simulation you just ran? Do you think your agents are behaving correctly?
Yes
No
I don't know
Are you sure? Do the agents in your simulation behave the same way as the agents in the expert simulation? Do the plots for your agent and environment properties match that for the expert?
I would agree. Remember, it is a good idea to read your Science Book and make sure you have all the needed information before you try correcting your agent behaviors.
Observe how the agents behave and how the agent and environment properties change over time in the simulation for your model, and then compare them with that of the expert. Are they the same?
Yes
No
Okay, I'll go and read first thing now.
I want to take a quiz and figure out what I should focus on while reading.
I know everything, I'm not planning on reading now.
Yes
No
I'm not sure
Well, I do not think so. I think you need to keep working on modeling your agent behaviors correctly.
Well, I think you need to keep working on modeling your agent behaviors correctly.
Oh, okay!
Okay, sounds good!
Take student to the quiz activity
Well, go ahead and try correcting your agent behaviors then. You can always consult the Science Book when you need to.
Oh, okay!
Remember, it is a good idea to read your Science Book and make sure you have all the needed information before you try correcting your agent behaviors.
Remember, it is a good idea to read your Science Book and make sure you have all the needed information before you try correcting your agent behaviors.
Agent prompt
Student prompt
Agent action

Figure 24. A scaffolding conversation tree asking students to read about incorrect agent behaviors

Our scaffolding approach is based on helping students with a task or strategy only we detect that they are persistently facing problems, instead of correcting them every time we detect a problem. Hence, the scaffolding reasoner maintains a frequency threshold for triggering each scaffold. In case of strategy scaffolds, the reasoner reads in the frequencies of suboptimal use of each strategy from the learner model and compares them with the corresponding strategy scaffold triggering frequency thresholds. When the suboptimal strategy use reaches the critical frequency, the scaffolding reasoner can choose to deliver scaffolds for the particular strategy if it fits with the stored priority or ordering of scaffolds. Similarly, the scaffolding reasoner stores frequency thresholds for triggering task based scaffolds for conceptual and computational modeling. It takes the frequency (say, '*n*') for triggering a particular modeling scaffold and looks at the history of a student's corresponding modeling skills to see if it can find '*n*' instances where the modeling skill has not shown improvement. Like we described in Section 6.4, students' modeling skills are captured locally in the learner model every time they perform a model assessment action. No improvement in modeling skill between model assessments is denoted by no improvement in any of the factors defining the modeling skill. For example, conceptual modeling skills comprise two factors: missing conceptual elements, and extra conceptual elements. Computational modeling skills, on the other hand, are defined by four factors: missing blocks, extra blocks, actions under correct conditions, and property values modified correctly (see Section 6.4 for details).

While we do not maintain a strict ordering between all of the task and strategy based scaffolds, we do maintain a priority list for situations where multiple scaffolds can be triggered simultaneously. For example, when a student performs a SA action where multiple compared agent behaviors have been modeled incorrectly, the '*Test-in-parts*' strategy scaffold gets triggered first if it meets its triggering requirements, followed by the '*Information acquisition prior to solution construction and assessment*' strategy scaffold. We first ensure that a student is not trying to compare too many incorrect behaviors simultaneously, because analyzing multiple errors at the same time may make it hard to pinpoint specific ones. When students test their model in parts, we provide scaffolds when we detect students have found incorrect agent behaviors, but they have not looked for information that will help them correct the error. If students have previously read about agent behaviors, but cannot correct incomplete or incorrect behaviors when testing in parts, we provide them with task-based model building scaffolds that hint at using information they have read to correct specific aspects of their model behaviors.

The model building scaffolding itself uses a top-down approach by providing conceptual modeling scaffolds as long as the ‘missing’ score for any of the conceptual components in the learner model is greater than zero. Specifically, the scaffolds point students to specific levels in the conceptual modeling hierarchy they need to focus on (starting with the set of entities, followed by the set of agent behaviors, and the sense-act properties for the behaviors) and suggest consulting relevant resource pages to acquire the required information for correct conceptual modeling. Once a student’s conceptual model contains all the elements contained in the expert conceptual model (it may still contain extra elements beyond those in the expert model), the coherence measure between the student’s conceptual sense-act models and computational models trigger the *Model-Build* strategy scaffolds, when applicable. The Model-Build scaffold leverages the visual feedback about conceptual-computational coherence provided by the system through the green or red coloring of the sense-act properties. The scaffold draws students’ attention to the properties colored red in their models, and reminds students that they can either delete the red properties from their conceptual model or add computational blocks that match the properties. Once there are no more sense-act properties colored red, the computational modeling scaffolds help point out whether there are missing or extra blocks in students’ computational models, or action blocks that have not been modeled under the correct set of conditions. The suggestions for rectifying the various types of computational modeling errors for different agent behaviors are similar – acquiring information about the agent behavior by carefully reading the science resources. A few examples of scaffolding dialogues are as follows: “*The fish-breathe behavior requires interaction of the fish with other entities. Have you considered all the entities in this science topic*?”; “*You have all the necessary blocks for the rollercoaster-update speed behavior, but are you sure that all the actions occur under the right set of conditions*?”; “*You have unused properties colored red in the fish-feed behavior. Do you want to use them in your program or do you want to delete the properties*?”

The *SC-IA* and *SA-IA* strategy scaffolds are mutually exclusive and do not share triggering conditions with any of the other strategy or task-based scaffolds; hence, they are triggered whenever their respective critical frequencies are reached. These scaffolds remind students about agents or agent behaviors recently modeled or assessed and ask if students are trying to gather information about any of them. Accordingly, students are provided suggestions on pages to read and reminded about how they can use the search feature to find relevant resource pages by themselves. All the other scaffolds are provided in the context of SA actions and start by asking students to evaluate the correctness of the simulations they just observed. They offer suggestions for testing a few agent behaviors at a time (*Test in parts*), or reading about the incorrectly modeled agent behaviors before trying to correct them (*IA-SA*).

While our scaffolds are triggered by possible sources of errors in students' modeling tasks, and they offer suggestions on how the students can debug and rectify these errors by efficiently integrating information available to them in the CTSiM environment, none of our scaffolds provide '*bottom-out-hints*' by telling students exactly what to correct in their model (Koedinger & Aleven, 2007). Also, though all our scaffolds are provided only when we detect students making multiple errors on a particular task, or multiple ineffective uses of a strategy, they often start with a positive message about students' previous successes in applying actions and strategies correctly.

In summary, modeling science topics using the CTSiM v2 environment is a complex process. The environment provides students with a number of supporting tools, but planning and using the tools effectively and combining information from them in a meaningful fashion is a non-trivial task. The large solution space implied by the open-ended nature of the environment and the variety of choices available to students make interpreting students' actions extremely difficult. Hence, we adopt a learner modeling and adaptive scaffolding approach that interprets students' actions in the CTSiM v2 system based on their performances on different system tasks and how they combine information from different tasks. We believe that this approach will help students become more effective learners and help them develop better modeling strategies.

CHAPTER 7

Experimental evaluation of the modified CTSiM environment with adaptive scaffolding

This chapter presents a research study designed to investigate the effectiveness of the CTSiM v2 environment with adaptive scaffolding. While experimental evaluation of each newly added tool and interface feature in the CTSiM v2 environment is possible, this dissertation focuses specifically on studying the effects of the adaptive scaffolding on students' science and CT learning, their modeling performance, and their learning behaviors. We discuss a recent controlled classroom study using CTSiM v2 with four sections of 6th grade students from a middle Tennessee middle school. Students, in this study, were divided into two approximately equal groups – a control group that used a version of the CTSiM v2 system without adaptive scaffolding, and an experimental group that used the full version of the CTSiM v2 system with adaptive scaffolding provided by Ms. Mendoza. Both groups had access to the new modeling and information acquisition interfaces, and model verification tools described in Chapter 5. Since these tools and interface features were available to all students at all times during this research study, they can be considered as 'blanket scaffolds' (Puntambekar and Hubscher, 2005), as opposed to the 'adaptive scaffolds,' which were only provided to students in the experimental group based on their observed modeling deficiencies.

7.1 Materials and Method

7.1.1 Research questions

We assess the effectiveness of the CTSiM v2 adaptive scaffolding framework by analyzing data generated from the controlled research study to answer the following six research questions. Research questions 1-5 are related to the effects of the adaptive scaffolds on students' learning gains, modeling performance and behaviors, while the 6th research question relates to studying the amount of adaptive scaffolding provided.

1. *What effects do the adaptive scaffolds have on students' (a) science and (b) CT learning?*
2. *How do the adaptive scaffolds impact students' conceptual and computational modeling performances?*

3. *What effects do the adaptive scaffolds have on students' abilities to transfer conceptual and computational modeling skills to model other science domains outside CTSiM?*
4. *How do the adaptive scaffolds impact students' modeling behaviors and use of effective strategies?*
5. *How do students' modeling performance and modeling behaviors including strategy use relate to their learning of the science concepts in the different modeling activities?*
6. *How do the requirements for different types of adaptive scaffolds vary during the course of the intervention, and do the scaffolds follow a desired 'fading' principle?*

7.1.2 Participants

We conducted the controlled classroom study with 98 6th grade students (average age = 11.5) from four sections of the same middle school in middle Tennessee. All students provided parental and personal consent to participate in the research study. The 6th grade science teachers assigned students from two of the sections to the control condition and students from the other two sections to the experimental condition. The teachers' looked at students' state level science scores from the previous year (5th grade TCAP scores) to make sure the two conditions were balanced in terms of their prior knowledge. The control condition (*n* = 46) used a version of the CTSiM system with no adaptive scaffolding provided by the mentor agent, Ms. Mendoza, and the experimental condition (*n* = 52) used the full version of the CTSiM system, i.e., the system used by the control condition, plus the learner modeling scheme and adaptive scaffolding based on the learner model provided by Ms. Mendoza.

7.1.3 Learning activities

The learning activities used in this research study with the CTSiM v2 environment are a slight modification of the initial learning activity progression used with the CTSiM v1 environment and described in Section 3.6. The learning activities still span topics in two curricular units – Kinematics followed by Ecology, but some of the initial activities now serve as introductory training activities. Students were not assessed on their training activities. In the study reported here, the CTSiM v2 learning progression comprises an introductory training activity and three primary modeling activities across two domains - Kinematics and Ecology.

Students start with an introductory shape drawing activity for training and practice. In this activity, students model a single tortoise agent and use simple CT concepts like the use of variables and the iteration construct to build closed shapes and spirals. The purpose of this activity is to explore the relations among distance, speed, and acceleration in relation to the shapes generated by the tortoise agent. The students start with an exercise where they program their tortoise to create a simple closed shape (square). This task is a collaborative activity in which the whole class participates. Students then move on to working individually, to construct other closed shapes (any regular polygon). All these activities represent constant speed movements of the tortoise. Then students are introduced to the concept of acceleration, and the effect of acceleration on speed is shown by shapes that represent growing and shrinking spirals. Since the shape drawing activities are treated as training and practice activities, students are allowed to seek help from and discuss solutions with their peers, teachers and researchers in the classroom.

In their first primary modeling activity, students progress to modeling a real-world phenomenon that introduces a more complex computational construct, like a conditional. Activity 1 models a roller coaster (RC) car moving along a pre-defined track with four segments - *up at constant speed* (pulled by a motor); *down* (free fall along the track under gravity); *flat* (constant speed on a flat track); and *up against gravity* (moving against gravity on a track). Section 3.6 and Figure 5 provide details. The plots generated depict how the speed, acceleration, and distance travelled by the RC car vary on the different segments of the track. In Activity 1 (as well as activities 2 and 3), students can compare the behavior of their model against an expert model behavior to verify whether their generated model is correct.

In Activities 2 and 3, students progress from modeling a domain with a single agent (the RC activity) to modeling ecological processes in a fish tank system, which represents a domain with multiple agents with multiple behaviors. These activities are the fish-tank macro and micro activities described in Section 3.6. In Activity 2, students build a *macro*-level, semi-stable model of a fish tank with two types of agents: fish and duckweed, and behaviors associated with the food chain, respiration, locomotion, and reproduction of these agents. Since the waste cycle is not modeled, the build-up of toxic fish waste creates a non-sustainable macro-model, where the fish and the duckweed gradually die off. In Activity 3, students address this lack of sustainability by introducing micro-level entities (agents), i.e., Nitrosomonas and Nitrobacter bacteria, which together support the waste cycle, by converting the ammonia in the fish waste to nutrients (nitrates) for the duckweed. The plots generated by the simulation models help students gain an aggregate level understanding of the different cycles in the fish tank ecosystem, and their role in establishing the interdependence and balance among the different agents in the system.

7.1.4 Study procedure

Students in both the control and experimental groups worked individually on the same learning activity progression, described in Section 7.1.3. The study was run daily over a span of three weeks during the students' science periods (one hour daily for each 6th grade section).

On Day 1, students took 3 paper-based tests that assessed their knowledge of (1) Kinematics, (2) Ecology, and (3) CT concepts. More details on the test questions are presented in Section 7.1.5. On day 2, students were introduced to agent based modeling concepts, and got a hands on introduction to the CTSiM v2 system. The whole class worked together on an introductory shape drawing activity – modeling a square. From Day 3, students worked individually in the CTSiM environment. On days 3 and 4, they worked on generating growing and shrinking spiral shapes, which emphasized the relations between distance, speed, and acceleration. Since the drawing tasks were considered a part of training and practice activities, students were allowed to help each other and seek help from their science teacher and the research team. From Day 5 students worked on the three primary modeling activities, and were not provided any individual help external to the system. Students worked on Activity 1, the Rollercoaster (RC) unit on days 5 and 6, after which they took paper-based post-tests for Kinematics and CT on Day 7. On days 8-12, students worked on modeling the ecological processes in a fish tank ecosystem. This model was built in two parts, as described earlier: modeling the macro (Activity 2) and micro (Activity 3) environments in the fish tank. Students took their Ecology and CT-final post-tests on Day 13. Finally, on Day 14, students worked on a paper-based transfer activity, where they had to build conceptual and block-based models for a new science topic described to them as part of the activity.

As students worked on the CTSiM system, all their actions on the system, along with the associated context were logged for future analysis. We analyzed the action logs to study the evolution of students' models for different learning activities, students' overall modeling performances at the end of the activities, the learning behaviors they exhibited, and the specifics of the feedback that was triggered and delivered by the mentor agent in the experimental condition.

One of our primary goals in this study was to assess the effectiveness of our adaptive scaffolds by comparing students in the control and experimental groups based on their (i) modeling performances and behaviors and (ii) science and CT learning gains, as measured by their actions in the system and performances on paper and pencil assessment artifacts outside the system, respectively.

7.1.5 CT and Science Assessments and the Transfer test

We designed paper based assessment instruments to evaluate students' science and CT learning and their abilities to transfer their conceptual and computational modeling skills to new scenarios. We measured students' learning gains for science content in the kinematics and ecology domains, and CT content as the differences between their pre- and post-test scores for the individual tests. The questions contained in the pre/post tests for (i) Kinematics, (ii) Ecology, and (iii) CT, as well as the paper-based transfer test for modeling skills are listed in Appendix B.

The Kinematics pre/post-test assessed whether students understood the concepts of acceleration, speed, and distance and their relations. The test required interpreting and generating speed-time and position-time graphs and generating diagrammatic representations to explain motion in a constant acceleration field. An example question asked students to diagrammatically represent the time trajectories of a ball dropped from the same height on the earth and the moon, and then to generate the corresponding speed-time graphs.

For the Ecology test, questions focused on students' understanding of the concepts of interdependence and balance in an ecosystem, and how a change in the population of one species in an ecosystem affects the other species. An example question asked was "*Your fish tank is currently healthy and in a stable state. Now, you decide to remove all traces of Nitrobacter bacteria from your fish tank. Would this affect a) Duckweed, b) Goldfish, c) Nitrosomonas bacteria? Explain your answer*."

CT skills were assessed by asking students to predict program segment outputs, and model scenarios using CT constructs. This tested students' abilities to develop meaningful algorithms using programmatic elements like conditionals, loops and variables. Simple questions tested use of a single CT construct, while modeling complex scenarios involved use of CT constructs, like conditionals and loops and domain-specific constructs.

For the paper-based learning transfer activity, students were provided with a detailed textual description of a wolf-sheep-grass ecosystem. Based on this description, students had to first build conceptual models for the agents in the system, much like the fish tank ecosystem. Then they had to build the block-based (computational) models of agent behaviors using computational and domain-specific modeling primitives that were specified in the question. This model building exercise was similar to the fish tank eco-system model they had built in CTSiM, except that the science domain was different and it was all done with pencil and paper. Therefore, unlike the CTSiM v2 environment, students did not have access to any of the online tools, nor did they get feedback from the system by simulating their model or from the mentor agent.

7.1.6 Log file analysis and Assessment metrics

Besides assessing students using paper-based assessment artifacts external to the CTSiM environment, we defined a set of assessment metrics to analyze the log data generated from this research study and use it to evaluate students' conceptual and computational modeling performances and behaviors, and use of desired and suboptimal learning strategies. The log data contained information about the different actions the students took in the CTSiM v2 environment along with the associated context in which the actions were taken, the models students built along with model revision history, and the scaffolds they received along with the ensuing conversations with Ms. Mendoza.

We assess a student's conceptual and computational modeling performance for an activity by defining distance metrics similar to those used online in the model comparator functions in the learner modeling module (see Section 6.4). The metrics specify the distances between the student's final models and the corresponding expert models, and a model distance of 0 implies that the student's model perfectly matches the expert model (no missing elements and no extraneous elements).

The conceptual model distance is calculated by normalizing the sum of the distances for the individual conceptual model components, i.e., agents, environment elements, agent properties, environment properties, agent behaviors, and sensed and acted-on properties for each agent behavior, by the size of the expert conceptual model. The distance metric is computed for any individual component by performing a simple set comparison between the elements of the component in a student's conceptual model and those contained in the corresponding expert conceptual model. The set difference provides the number of ‘missing’ and ‘extra’ elements for the component (see Equations 1 and 2). The ‘missing’ measure for a component counts the number of elements of that component that are present in the expert model, but missing in the student model. Similarly, the ‘extra’ measure for a component counts the number of elements of that component that are present in the student model, but not present in the expert conceptual model. The ‘distance’ measure (see Equation 3), computed as the sum of the ‘missing’ and ‘extra’ measures across all conceptual model components, is normalized by the size of the expert conceptual model (i.e., the sum of the number of elements of each type of conceptual component) to make the ‘distance’ measure independent of the size of the expert conceptual model.

$$conceptual\ missing\ score = \sum_{each\ conceptual\ component} |expert - user| \tag{1}$$

$$conceptual\ extra\ score = \sum_{each\ conceptual\ component} |user - expert| \tag{2}$$

$$weighted\ conceptual\ distance = \frac{conceptual\ missing\ score + conceptual\ extra\ score}{\sum_{each\ conceptual\ component} |expert|} \tag{3}$$

The computational model distance is calculated by computing separate ‘correctness’ and ‘incorrectness’ measures for a student’s computational model (see Equations 4 and 5), and measuring the vector distance (see Equation 6) between the two-dimensional vector (correctness, incorrectness) to the target vector (1,0) (Basu et. al., 2014; Basu, Kinnebrew & Biswas, 2014). The total correctness and incorrectness measures are calculated by combining the respective measures from the individual agent behaviors using a weighted average based on the size of each behavior’s expert model. The correctness measure for a single agent behavior is computed as the size of the intersection of the collection of visual primitives used in the student and expert models for the behavior. Similarly, the incorrectness measure for an agent behavior is computed as the number of extra primitives in the student computational model as compared to the expert model. In Section 6.4, while describing the computational model comparator function of the learner modeling module, we detailed how we went beyond a mere block-based comparison of student and expert computational models and accounted for various false positives and false negatives. We follow a similar approach in our offline log-data based analysis of students’ computational modeling performance. When we detect instances of false positives (for example, actions under incorrect conditions), we do not consider corresponding blocks (action blocks in this example) as part of the user model, thus adjusting the correctness score. Similarly, in case of false negatives, we match the blocks used by students to semantically similar expert model blocks, thus adjusting the correctness and incorrectness scores.

$$weightedAverageCorrectness = \frac{\sum_{each\ procedure} |user \cap expert|}{\sum_{each\ procedure} |expert|} \quad (4)$$

$$weightedAverageIncorrectness = \frac{\sum_{each\ procedure}(|user| - |user \cap expert|)}{\sum_{each\ procedure} |expert|} \quad (5)$$

$$distance = \sqrt{incorrectness^2 + (correctness - 1)^2} \quad (6)$$

Besides using log data to study students' modeling performances for each learning activity, we also study students' modeling behaviors for each activity in terms of the following: (i) conceptual and computational model evolution over the course of the activity, (ii) how the conceptual and computational representations are combined during the activity, and (iii) frequency of use of the desired and suboptimal strategies *S1-S5* described in Section 6.3.

With respect to both conceptual and computational modeling, we describe students' model evolution during an activity by calculating the model distances at each model revision (actions performed as part of the SC task defined in the CTSiM task model, see Section 6.3) and then characterizing the model evolution using 3 metrics:

(1) *Effectiveness*- the proportion of model edits during the activity that bring the model closer to the expert model for the activity;

(2) *Slope* – the rate and direction of change in the model distance over time as students build their models for the activity; and

(3) *Consistency* – How closely the model distance evolution over time in an activity matches a linear trend.

We assess students' modeling behavior in terms of how they combined the linked conceptual and computational modeling representations to build their models (Basu, Biswas, & Kinnebrew, 2016) using the following metrics:

(1) *Number of switches* - We look at conceptual and computational activity chunks (successive actions in one type of representation) and use the total number of chunks as a measure of how many times a student switched between the two representations;

(2) *Average conceptual chunk size* - The average number of consecutive conceptual modeling actions taken before a switch is made to the computational modeling representation;

(3) *Average computational chunk size* - The average number of consecutive computational modeling actions taken before a switch is made to the conceptual modeling representation*;*

(4) Weighted ratio of chunk sizes – The ratio of conceptual and computational chunk sizes divided by the ratio of the sizes of the conceptual and computational expert models; and

(5) *Coherence* - the fraction of conceptual edits that were followed by *related* (*coherent*) computational edits at some future time.

Further, in addition to assessing students' modeling behaviors in terms of their model evolutions for each learning activity and how they integrate the conceptual and computational modeling representations, we characterize students' modeling behaviors in terms of their frequency of use of the desired strategies S1-S5 (see Section 6.3), and their suboptimal variants.

7.2 Results

We analyzed students' responses on the paper-based assessment artifacts and the data logged as they used the CTSiM v2 environment to answer the six research questions presented in Section 7.1.1. The effectiveness of our adaptive scaffolding framework is demonstrated by comparing students in the control and experimental groups based on their learning gains, modeling performances and behaviors, and abilities to transfer modeling skills to new scenarios.

7.2.1 Science and CT learning gains

To answer our 1st research question, we analyzed the impact of our task and strategy based scaffolds on students' overall science and CT learning, as measured by the differences between their performance on the corresponding pre- and post-tests. Table 7 reports the pre- and post-scores and pre-post gains for students in both conditions for the paper-based science (Kinematics and Ecology) assessments as well as for the CT assessment. The CT post-test score refers to scores on the CT tests administered on Day 13 of the study at the end of the ecology unit.

Students in both conditions showed significant pre-post learning gains for kinematics and ecology science content, as well as CT concepts and skills. However, the gains and effect sizes (*Cohen's d*) were higher in each case for students in the experimental group (*n* = 52) compared to those in the control group (*n* = 46). Also, though we made every effort to balance the control and the experimental groups using scores from the previous year's standardized tests (TCAP scores), we notice that students in the experimental group had higher pre-test scores, hence we computed ANCOVAs comparing the gains between control and experimental conditions taking pre-test scores as covariates. Factoring out the effect of initial knowledge differences implied by the pre-test scores, we found significant differences in science learning gains between the two conditions with medium to high effect sizes: kinematics gains ($F = 18.91, p < 0.0001, \eta p^2 = 0.17$) and ecology gains ($F = 52.29, p < 0.0001, \eta p^2 = 0.36$). Similarly, we factored out CT pre-test effects to find a significant effect of condition on CT learning gains ($F = 40.69, p < 0.0001, \eta p^2 = 0.31$). We also assessed students' performances on the first CT post-test taken at the end of kinematics unit, and found that students in the experimental group showed higher learning gains from the pre-test to the first post-test ($F = 18.16, p < 0.0001, \eta p^2 = 0.16$) , and gained further from the intermediate to the final CT post-test administered at the end of the ecology unit ($F = 18.85, p < 0.0001, \eta p^2 = 0.17$). Therefore, in answering our first research question we conclude that the adaptive scaffolding described in Sections 6.4 and 6.5 helped students achieve higher science and CT learning gains.

Table 7. Science and CT learning gains for students in the control and experimental conditions

		Pre	**Post**	**Pre-to-post gains**	**Pre-to-post *p*-value**	**Pre-to-post *Cohen's d***
Kinematics (max = 45)	Control	12.52 (6.32)	15.55 (5.72)	3.03 (4.78)	<0.0001	0.55
	Experimental	16.65 (6.61)	22.38 (6.39)	5.72 (5.62)	<0.0001	0.88
Ecology (max = 39.5)	Control	7.40 (3.90)	16.19 (8.35)	8.78 (7.17)	<0.0001	1.35
	Experimental	9.39 (4.47)	27.91 (6.70)	18.53 (6.31)	<0.0001	3.25
CT (max = 60)	Control	16.49 (5.68)	22.53 (5.70)	6.04 (5.44)	<0.0001	1.06
	Experimental	22.72 (7.68)	32.24 (5.86)	9.52 (5.23)	<0.0001	1.39

We also noticed that the average of the control condition's post-test scores turned out to be lower than the experimental condition's average pre-test scores for the science and CT units. In order to probe deeper into why this occurred, we calculated the average pre- and post-test scores for students in each of the four 6th grade sections to check whether any section was an outlier in terms of pretest performance. Our results (see Table 8) showed that both sections assigned to the experimental condition performed at par on the pre-tests, and the two sections assigned to the control condition performed similarly on the pre-tests. Therefore, we were left to answer the question: did the students in the experimental condition learn more because they already knew more than the students in the control condition?

Table 8. Science and CT learning gains by section

		Pre	Post	Pre-to-post *p*-value
Kinematics (max = 45)	Section1 – Control (n=22)	13.64 (6.9)	16.41 (6.2)	<0.01
	Section3 – Control (n=24)	11.5 (5.7)	14.77 (5.24)	<0.01
	Section 2 – Experimental (n=26)	17.12 (6.6)	21.4 (6.6)	<0.0005
	Section 4 –Experimental (n=26)	16.19 (6.7)	23.35 (6.13)	<0.0001
Ecology (max = 39.5)	Section1 – Control (n=22)	8.25 (4.6)	18.75 (9.4)	<0.0001
	Section3 –Control (n=24)	6.63 (3.0)	13.83 (6.7)	<0.0001
	Section 2 – Experimental (n=26)	9.0 (5.4)	25.58 (7.5)	<0.0001
	Section 4 –Experimental (n=26)	9.77 (3.4)	28.25 (5.9)	<0.0001
CT (max = 60)	Section1 – Control (n=22)	14.89 (5.7)	22.55 (6.4)	<0.0001
	Section3 –Control (n=24)	17.96 (5.4)	22.52 (5.1)	<0.0001
	Section 2 – Experimental (n=26)	22.87 (8.6)	32.77 (6.6)	<0.0001
	Section 4 –Experimental (n=26)	22.58 (6.8)	31.71 (5.2)	<0.0001

To investigate the question above, we divided students in each condition into two groups ('Low pre scores' and 'High pre scores') based on their pre-test performances using the median score as the divider, and compared learning for students who started with low pre-test scores against those who started with high pre-test scores. Table 9 shows the learning gains for both the groups in each condition. The effect sizes reported in Table 9 and the slope of the plots in Figure 25 show that the students in both control and experimental conditions who start with lower pre test scores have higher pre-to-post learning gains compared to students in the same condition who start with high pre-test scores. This result holds good for both science and CT learning, and demonstrates that CTSiM is not biased toward the students who are more knowledgeable initially. Instead, students who initially have low knowledge tend to learn more, and this helps reduce the gap between students who start with differing levels of prior knowledge.

Also, we observe that the experimental group students with low pre score show higher learning gains than the control group students with low pre scores, and the experimental group students with high pre-test scores show higher learning gains than the control group students with high pre-test scores. In addition, we find that experimental group students with low pre-test scores generally have pre-test scores much lower than that of control group students with high pre-test scores. However, by the post-test, this 'experimental – low pre score' group not only catches up, but also performs better than the 'control-high pre score' group.

Table 9. Gains by low and high pre-test performance for each condition

		Pre	Post	Pre-to-post *p*-value	Cohen's d Effect size
Kinematics (max = 45)	Control – Low pre scores(n=23)	7.63 (2.9)	12.5 (4.8)	<0.0001	1.23
	Control – High pre scores(n=23)	17.41 (4.8)	18.61 (4.9)	>0.05	0.25
	Experimental – Low pre scores (n=26)	11.38 (3.8)	18.98 (6.0)	<0.0001	1.51
	Experimental – High pre scores (n=26)	21.92 (4.1)	25.77 (4.8)	<0.0001	0.86
Ecology (max = 39.5)	Control – Low pre scores(n=20)	4.80 (1.2)	12.98 (6.5)	<0.0001	1.75
	Control – High pre scores(n=21)	10.09 (4.3)	19.41 (9.4)	<0.0001	1.27
	Experimental – Low pre scores (n=22)	5.64 (1.6)	25.93 (7.0)	<0.0001	4.0
	Experimental – High pre scores (n=25)	12.86 (3.8)	29.66 (5.8)	<0.0001	3.43
CT (max = 60)	Control – Low pre scores(n=23)	11.81 (2.8)	19.59 (4.8)	<0.0001	1.98
	Control – High pre scores(n=23)	21.17 (3.5)	25.48 (5.1)	<0.001	0.99
	Experimental – Low pre scores (n=26)	16.39 (4.7)	28.7 (4.8)	<0.0001	2.59
	Experimental – High pre scores (n=26)	29.06 (3.8)	35.77 (4.6)	<0.0001	1.59

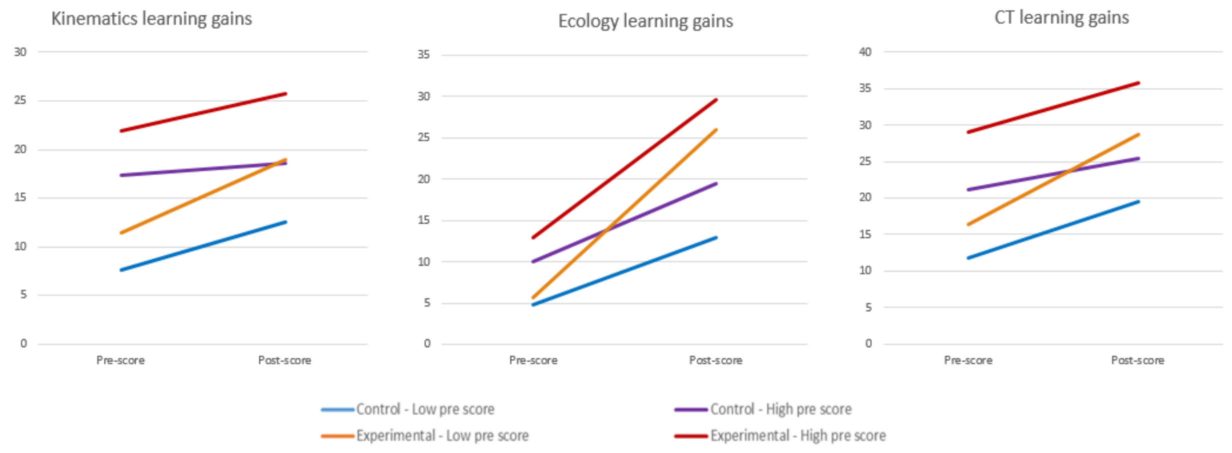


Figure 25. Science and CT learning gains by condition and initial pre-test scores

7.2.2 Modeling performance

In order to address our 2nd research question, we assess the effectiveness of our adaptive scaffolding framework by comparing the model building performance of students in the control (n = 46) and the experimental (n = 52) groups. Modeling performance for an activity is measured in terms of the accuracy of students' final conceptual and computational models using 'distance' metrics outlined in Section 7.1.6. Table 10 and Figure 26 illustrate students' conceptual and computational modeling performance, where a lower distance score indicates better model performance.

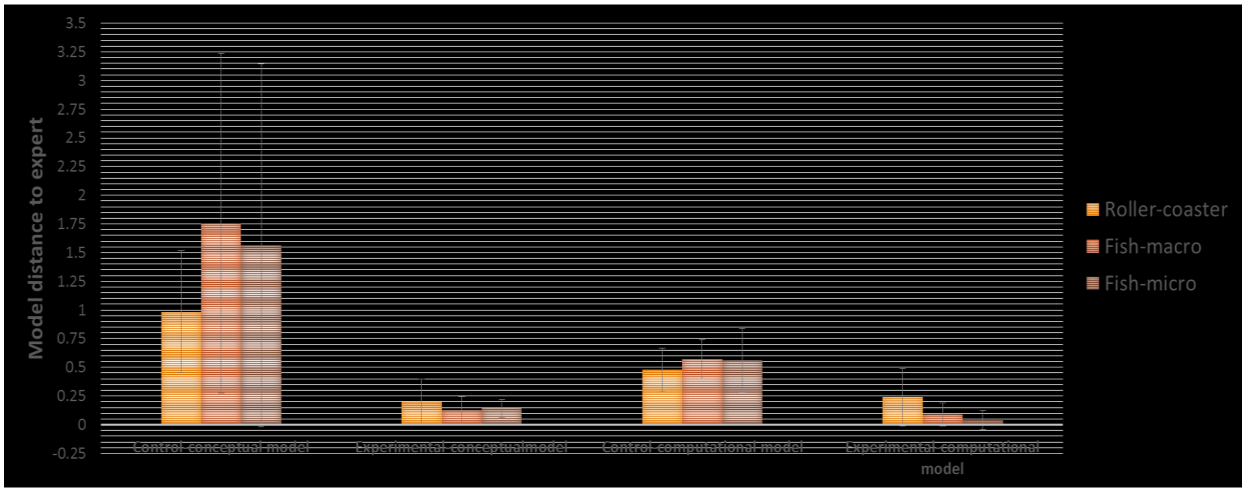


Figure 26. Modeling performance across conditions

Figure 26 shows that students in the experimental condition built more accurate conceptual models for the Rollercoaster, Fish-macro, and Fish-micro activities (the final model distance scores were significantly lower) compared to students in the control condition who did not receive any scaffolding from Ms. Mendoza. Breaking down the aggregate distance scores, Table 10 shows that the two component scores of 'missing' and 'extra' constructs were also significantly lower for the experimental condition, implying that the experimental group's models included more of the conceptual model elements from the expert model (significantly lower 'missing' score) and fewer redundant and incorrect conceptual elements (significantly lower 'extra' score), compared to the control group's models.

Similarly, students in the experimental condition built more accurate computational models compared to students in the control condition (the differences in final model distances for the two groups were statistically significant) for the Rollercoaster, Fish-macro and Fish-micro modeling activities. Similar to conceptual modeling, the experimental group's computational models included more of the expert model elements (significantly higher correctness score) and fewer redundant and incorrect computational elements (significantly lower incorrect score) than the control group's models.

In summary, the answer to our 2nd research question is that the adaptive scaffolding through conversational feedback provided to students in the experimental condition resulted in significantly better conceptual and computational modeling task performance compared to students in the control group.

Table 10. A comparison of modeling performances across conditions (Note: *$p < 0.05$, **$p < 0.001$, ***$p < 0.0001$)

			Rollercoaster	Fish-macro	Fish-micro
Final conceptual model accuracy	Missing score	Control	0.09 (0.12)	0.23 (0.14)	0.19 (0.16)
		Experimental	0. 02 (0.05)*	0.04 (0.05)***	0.04 (0.02)***
	Extra score	Control	0.90 (0.59)	1.53 (1.52)	1.38 (1.60)
		Experimental	0.17 (0.18)***	0.09 (0.09)***	0.10 (0.07)***
	Distance score	Control	0.99 (0.53)	1.76 (1.48)	1.56 (1.58)
		Experimental	0.20 (0.20)***	0.13 (0.13)***	0.14 (0.08)***
Final computational model accuracy	Correctness score	Control	0.66 (0.23)	0.48 (0.21)	0.53 (0.27)
		Experimental	0.85 (0.21)***	0.93 (0.1)***	0.97 (0.07)***
	Incorrectness score	Control	0.24 (0.21)	0.15 (0.13)	0.21 (0.23)
		Experimental	0.15 (0.18)*	0.04 (0.03)***	0.02 (0.05)***
	Distance score	Control	0.48 (0.19)	0.57 (0.17)	0.56 (0.28)
		Experimental	0.24 (0.25)***	0.09 (0.1)***	0.04 (0.08)***

7.2.3 Transfer of modeling skills

Next, we analyzed students' performances on the transfer task where they were provided with a detailed description of a wolf-sheep-grass ecosystem and were asked to (i) build a conceptual model using an agent based sense-act framework, similar to the one students used while working in the CTSiM environment, and (ii) build a computational model using domain-specific and domain-general computational primitives provided in the question. We scored students' conceptual and block-based computational models of the wolf-sheep-grass ecosystem separately, and report our results in Table 11. The conceptual modeling score was determined by the number of conceptual elements (agents, environment elements, properties, behaviors, sense-act properties) required to correctly and completely describe the given wolf-sheep-grass ecosystem, minus the number of redundant conceptual elements. The computational modeling score also took into account required and extra blocks, as well as whether actions occurred under the correct set of conditions. We found that students in the experimental condition were able to apply their modeling skills better to the wolf-sheep-grass scenario, and built significantly more accurate conceptual and computational models compared to students in the control condition. This answers our 3rd research question.

Table 11. A comparison of learning transfer between conditions

		Control	**Experimental**	***p*-value**	***Cohen's d***
Conceptual modeling score	Conceptual entities (max = 5)	4.66 (0.79)	4.92 (0.39)	<0.05	0.43
	Conceptual sense-act (max = 41)	11.54 (5.29)	20.93 (6.70)	<0.001	1.56
	Total score (max=46)	16.21 (5.45)	25.86 (6.73)	<0.001	1.58
Computational modeling score (max=48)		17.33 (9.23)	30.50 (8.98)	<0.001	1.46
Total transfer test score (max=94)		33.53 (13.80)	53.36 (14.49)	<0.001	1.63

7.2.4 Modeling behaviors

We assessed students' modeling behaviors in each activity based on their: (i) model evolutions over time during the course of the activity, (ii) integration of the conceptual and computational modeling representations during the activity, and (iii) frequency of use of desired strategies S1-S5 and their suboptimal variants.

In addition to building more accurate final models, the experimental group's progress towards building their final models was significantly better than that of the control group (see Table 12). For example, the experimental group demonstrated better conceptual modeling behavior as evidenced by three metrics: (1) higher percentage of effective (i.e., correct) conceptual edits in all three activities; (2) conceptual model accuracy improved with time in each activity, i.e., the slope for model distance over time was negative, whereas the distance slope for the control group was positive. (The control group kept adding unnecessary elements to their models, and their conceptual models became more inaccurate in each activity as time progressed); and (3) modeling consistency was higher for the experimental group in the fish-micro unit. Also, the experimental group's computational model progressions within each unit were more consistent and improved more rapidly. Both conditions had negative computational model evolution slopes, i.e., their model accuracy improved over time in each of the activities. However, the rate of improvement was significantly higher for the experimental group in all the activities.

Table 12. A comparison of model evolutions across conditions (Note: *$p < 0.05$, **$p < 0.001$, ***$p < 0.0001$)

			Rollercoaster	Fish-macro	Fish-micro
Conceptual model progression	Edit Effectiveness	Control	0.497 (0.060)	0.445 (0.101)	0.483 (0.164)
		Experimental	0.567 (0.038)***	0.592 (0.044)***	0.676 (0.062)***
	Model evolution slope	Control	0.005 (0.007)	0.003 (0.003)	0.002 (0.006)
		Experimental	-0.003 (0.003)***	-0.002 (0.002)***	-0.005 (0.004)***
	Model evolution consistency	Control	0.334 (0.291)	0.500 (0.336)	0.585 (0.340)
		Experimental	0.304 (0.221)	0.591 (0.310)	0.796 (0.225)**
Computational model progression	Edit effectiveness	Control	.43 (.09)	.47 (.07)	.55 (.12)
		Experimental	.43 (.08)	.58 (.08)***	.69 (.11)***
	Model evolution slope	Control	-.004 (.004)	-.002 (.001)	-.005 (.004)
		Experimental	-.006 (.005)*	-.004 (.002)***	-.009 (.004)***
	Model evolution consistency	Control	.41 (.31)	.78 (.21)	.78 (.24)
		Experimental	.6 (.25)**	.95 (.04)**	.95 (.05)**

Table 13. A comparison of modeling behaviors across conditions with respect to combining the conceptual and computational representations
(Note: *$p < 0.05$, **$p < 0.01$, ***$p < 0.001$, ****$p < 0.0001$)

		Rollercoaster	**Fish-macro**	**Fish-micro**
Number of conceptual/ computational chunks	Control	20.13 (10.25)	55.02 (26.09)	30.07 (15.2)
	Experimental	33.23 (11.57)****	93.52 (30.11)****	56.17 (13.56)****
Average size of conceptual chunks	Control	10.24 (4.48)	18.54 (13.01)	20.29 (16.21)
	Experimental	8.24 (2.44)**	8.12 (3.33)****	5.65 (1.6)****
Average size of computational chunks	Control	16.72 (18.08)	8.82 (4.14)	7.2 (4.47)
	Experimental	7.92 (2.78)***	5.11 (1.25)****	4.2 (1.26)****
Normalized ratio of conceptual to computational chunk sizes	Control	0.83 (0.5)	2.66 (1.6)	2.73 (1.7)
	Experimental	1.1 (0.52)**	2.02 (0.87)*	1.38 (0.42)****
Fraction of conceptual edits with coherent computational edits	Control	.28 (.07)	.17 (.08)	.25 (.15)
	Experimental	.3 (.08)	.33 (.11)****	.56 (.12)****

Besides comparing students' use of the conceptual and computational representations separately, we also compared students' modeling behaviors with respect to how they combined the two representations using the metrics described in Section 7.1.6. Table 13 shows that the average sizes (number of edits) of the conceptual and computational chunks was significantly smaller for students in the experimental condition, while the number of switches between the conceptual and computational modeling representations was significantly higher for the experimental group. These results indicate that students in the experimental condition were better at decomposing their modeling tasks into smaller and more manageable chunks, and they switched frequently to take advantage of the coupled representations (Basu, Biswas & Kinnebrew, 2016; Basu, Kinnebrew, & Biswas, in review). This difference was consistent and statistically significant across all three modeling activities, but the disparity in both conceptual and computational chunk sizes became more pronounced in the later activities. In addition to decomposing their modeling tasks better, the students in the experimental condition also demonstrated a better understanding of the relations between the two levels of modeling abstractions, as evidenced by a higher number of conceptual edits followed by coherent computational edits.

The normalized ratio of conceptual and computational chunk sizes provides a complementary measure with respect to integration of the modeling representations. For each of the modeling activities, we noticed a significant difference in this normalized ratio between students in the two conditions, and found that the ratio was always closer to 1 for students in the experimental condition. A normalized chunk size ratio of 1 for an activity implies that it is equal to the ratio of the number of conceptual and computational elements in the expert model for the activity. This ratio increased from the RC to the fish-macro activity for both conditions, implying that students' conceptual edits increased as compared to their computational edits with respect to the expert models. However, the increase was significantly greater for the control group. Perhaps, the complexity of the fish-macro activity resulted in students spending more effort (i.e., more edits, because they made more errors) in conceptualizing the models (multiple entities, their properties, and behaviors) than in the RC unit. For the experimental condition, the normalized ratio decreased from the macro to the micro unit, implying that students had to spend less effort in conceptualizing the domain model. However, the ratio increased further from macro to micro activities for the control group.

Finally, we computed the impact of the adaptive scaffolding on students' modeling behaviors by comparing students in the control and experimental groups in terms of their effective and suboptimal strategy usage. Table 14 presents the average number of times each of the five strategies was used in each modeling activity, as well as the percentage of students who used the strategy at least once in each activity. We note two general trends in the effective use of all the strategies across the three modeling activities: (1) the fraction of students in the experimental group who used the strategies effectively was always greater than or equal to the fraction that used the same strategy effectively in the control group, and (2) the average effective uses of the strategies was also higher in the experimental group. As shown in Table 14, a number of the differences between average uses of strategies in the two conditions were statistically significant at different confidence levels. While most of the differences had low to medium effect sizes (*Cohen's d* in the range of 0.2 to 0.7), the differences in use of the coherent *Model-Build* strategy had much larger effect sizes in all three modeling activities (*Cohen's d* in the range of 1.36 to 1.75).

Table 14. A comparison of the use of desired strategies across conditions
(Note: *$p < 0.05$, **$p < 0.01$, ***$p < 0.001$)

		RC		**Fish-macro**		**Fish-micro**	
Strategy		Fraction of students	Mean (s.d.)	Fraction of students	Mean (s.d.)	Fraction of students	Mean (s.d.)
S1. SC action followed by relevant science reads	C	37%	1.33 (2.99)	54%	2.43 (4.8)	70%	1.93 (2.05)
	E	63%	2.23 (4.71)	83%	4.75 (4.97)*	85%	3.4 (4.51)*
S2. SA actions followed by relevant science reads	C	4%	0.07 (0.33)	26%	0.76 (1.66)	26%	0.85 (9.31)
	E	38%	1.37 (2.69)**	44%	1.66 (2.29)*	44%	1.06 (0.24)
S3. Fraction of assessed behaviors that were read about before being assessed	C	80%	.73 (.42)	93%	.5 (.33)	83%	0.89 (0.27)
	E	92%	.86 (.28)	96%	.77 (.32)***	100%	0.96 (0.16)
S4. Number of partial-model comparisons	C	0%	na	48%	2.65 (5.79)	15%	0.57 (1.98)
	E	0%	na	58%	5.42 (7.16)*	19%	1.97 (3.22)*
S5. Fraction of sense-act properties removed or followed by a coherent computational edit	C	100%	0.67 (0.27)	100%	0.69 (0.31)	98%	0.59(0.31)
	E	100%	0.97 (0.1)***	100%	0.99 (0.03)***	100%	0.98 (0.06)***

We also performed more fine grained analysis of effects of the scaffolds on effective uses of strategies by counting the number of effective uses before and after feedback instances. Our results show a general trend for students who needed scaffolding, their effective uses of strategies became more frequent as they received feedback for their suboptimal uses (Basu & Biswas, 2016). For example, for *S4* (the test-in-parts strategy) in the fish-macro unit, 10 of the 52 experimental group students never received feedback on *S4* and made 0.8 (1.5) effective uses of *S4* on an average. 15 students received feedback exactly once, and made an average of 2.0 (4.7) partial model comparisons before receiving feedback, which increased to 2.73 (6.24) after receiving feedback. The other 27 students received feedback on *S4* two or more times; they used *S4* an average of 0.93 (2.4) times before receiving any feedback, 1.93 (4.2) times between the first and second feedback instances, and 4.7 (7.43) times after receiving feedback twice.

Table 15. Comparing suboptimal uses of strategies in terms of feedback received or would be received (Note: *$p < 0.05$, **$p < 0.005$, ***$p < 0.0001$)

		RC			**Fish-macro**			**Fish-micro**		
		n	Min-max	Mean(s.d.)	*n*	Min-max	Mean(s.d.)	*n*	Min-max	Mean(s.d.)
S1: SC-IA strategy	C	4	0-1	0.09 (0.28)	19	0-28	1.93 (4.46)	15	0-7	1.13 (1.98)
	E	3	0-1	0.06 (0.2)	22	0-4	0.69 (1.02)	8	0-1	0.15 (0.36)**
S2: SA-IA strategy	C	0	0	0(0)	0	0	0(0)	3	0-10	0.28 (1.5)
	E	0	0	0(0)	0	0	0(0)	0	0	0(0)
S3: IA-SC/SA strategy	C	16	0-57	8.43 (15.8)	41	0-62	18.8 (15.7)	8	0-14	1.11 (2.94)
	E	18	0-15	1.37 (3.11)**	19	0-14	1.81 (3.27)***	4	0-3	0.13 (0.52)*
S4: Test-in-parts strategy	C	0	0	0(0)	46	1-26	9.57 (6.77)	37	0-30	3.85 (5.27)
	E	0	0	0(0)	42	0-9	2.23 (2.13)***	23	0-6	0.83 (1.26)***
S5: Model-Build strategy	C	41	0-32	7.17 (6.19)	45	0-130	34.83 (28.87)	36	0-150	18.85 (26.95)
	E	32	0-8	1.79 (2.17)***	35	0-10	2.04 (2.43)***	30	0-10	1.33 (1.82)***

We also studied the effect of our adaptive scaffolds on students' suboptimal uses of strategies. Since the strategy oriented scaffolds were triggered based on the suboptimal strategy uses, we counted the feedback received in the experimental group and calculated the feedback that would be received by the control group. For each of the five types of strategy feedback, Table 15 provides for each activity: (1) *n*, which represents the number of students who receive the feedback at least once in the activity, (2) *min-max*, where *min* represents the lowest number of times feedback is received by a student in the group during the activity (correspondingly, *max* represents the highest number of feedback instances received by a student in that group), and (3) *mean (s.d.)* represent the average number of times (and standard deviation) the feedback was received during the activity. We see that the students in the experimental group need significantly lower amount of strategy feedback than the control group would have needed, especially for the *Model-Build* strategy, the *test-in-parts* strategy, and the *IA-SC/SA* strategy, implying that the adaptive scaffolds helped improve effective uses of the strategies, and reduced their suboptimal uses.

In summary, the results imply that the adaptive scaffolding had a strong effect on students' modeling behaviors, and helped improve students' use and integration of the conceptual and computational modeling representations, and effective strategy usage.

7.2.5 Relations between modeling performances and behaviors, and science learning

To investigate our 5th research question, we analyzed the correlations between (a) the modeling performances and behaviors and strategy use for each activity, and (b) students' post-test scores for the corresponding science domain.

First, we correlated students' science post-test scores with their modeling performances, model evolution metrics, and how they integrated the conceptual and computational representations (see Table 16). We did not find any significant correlations between students' modeling measures in the Rollercoaster activity and their Kinematics post-test performances. A likely reason is that the RC conceptual representation, with a single agent type, did not provide a lot of scaffolding for designing the corresponding computational models. Therefore, the benefits of the linked representation were not as apparent. Besides, the students may not have become proficient with the representations in Activity 1, therefore, modeling alone did not perhaps help the students to better understand domain knowledge. However, Table 16 shows that students' modeling metrics in the fish-macro and fish-micro activities were correlated with their ecology post-test scores.

We find that the macro and micro final model distances were negatively correlated with ecology post-test scores, implying that lower distances to expert models or more accurate student models were associated with higher post test scores. For students in the experimental group, the correlations between their Ecology post-test scores and their computational performance measures for the fish-micro unit do not reach significance, possibly due to the homogeneity of their final model distances in the micro unit. Also, in the Ecology units, model evolution slopes, especially for conceptual modeling, appears to be a strong predictor of science post-test scores. The more negative the slope (i.e., faster the progression towards a more accurate final model), the higher the Ecology post-test score.

Table 16. Correlations of modeling performances and behaviors with science learning
(Note: *$p < 0.05$, **$p < 0.01$, ***$p < 0.001$, ****$p < 0.0001$)

Correlations with Kinematics post test scores				
	Control		**Experimental**	
	Conceptual	Computational	Conceptual	Computational
RC final distance	-0.16	-0.16	-0.208	-0.151
RC effectiveness	-0.17	-0.18	0.258	-0.013
RC slope	-0.09	-0.2	0.008	-0.111
RC consistency	-0.07	0.1	-0.158	0.096
RC number of chunks	0.06		0.028	
RC ratio of chunk sizes	0.02		0.055	
RC coherent edits	0.06		-0.005	
Correlations with Ecology post test scores				
Macro final distance	-0.35*	-0.56***	-0.476**	-0.393**
Macro effectiveness	-0.02	0.09	0.334*	0.225
Macro slope	-0.43**	-0.12	-0.204	-0.265
Macro consistency	-0.24	0.18	0.31162*	0.264
Macro number of chunks	0.61****		0.076	
Macro ratio of chunk sizes	-0.25		0.27	
Macro coherent edits	0.48***		0.293*	
Micro final distance	-0.43**	-0.62****	-0.393**	-0.182
Micro effectiveness	0.03	0.38**	0.275*	0.131
Micro slope	-0.53***	-0.46**	-0.286*	-0.155
Micro consistency	-0.08	0.22	0.281*	0.104
Micro number of chunks	0.51***		0.125	
Micro ratio of chunk sizes	-0.22		0.031	
Micro coherent edits	0.63****		0.275*	

In terms of linked representation integration metrics, we found that a higher number of chunks (greater number of switches between the conceptual and computational representations) and lower average chunk sizes were correlated with higher post-test scores for students in both conditions, suggesting that effective coordination between the linked modeling representations appeared to have a positive effect on science learning. Specifically, decomposing the modeling task and going back-and-forth between representations in relatively small sized chunks appeared to be useful behaviors that supported greater learning. Also, greater coherence between the conceptual and computational modeling representations in the fish-macro and fish-micro units strongly correlated with higher Ecology post-test scores for students in both conditions.

Furthermore, we also analyzed the correlations between students' strategy use in each activity and their post test scores in the corresponding science domain. While we found use of certain strategies to be significantly positively correlated to learning in particular units (for example, the Model-Build strategy in fish-macro and fish-micro activities), we did not generally find use of an individual strategy to be correlated with learning across all activities or across conditions. This result speaks for the importance of using a combination of the strategies for efficiently integrating the different CTSiM tasks and sub-tasks, since we have found that the experimental group students who displayed a better overall usage of the desired strategies also displayed higher learning gains.

7.2.6 Fading effect of scaffolds across modeling activities

Finally, to answer our 6th research question, we studied how often students in the experimental group received scaffolding, and how the scaffolding frequency varied across the three modeling activities. Figure 27 illustrates how the strategy-based and task-oriented feedback received by students in the experimental condition varied across learning activities.

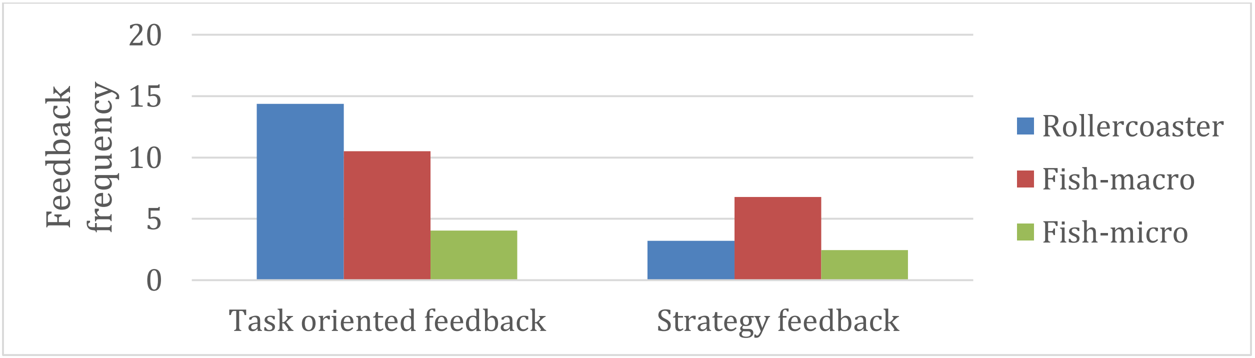


Figure 27. Frequency of feedback received across learning activities

Table 17 shows the feedback received for different strategies and different aspects of the modeling task in each activity. For each type of task-based and strategy-oriented feedback, Table 17 provides 3 values for each activity: (1) n represents the number of students who received the feedback at least once in the activity, (2) range represents the lowest and highest number of times the feedback was received by any student during the activity, and (3) mean (s.d.) represent the average number of times the feedback was received during the activity along with its standard deviation value.

Table 17. Variation of frequency and types of scaffolds required across modeling activities

			RC			**Fish-macro**			**Fish-micro**		
			n	Range	Mean (s.d.)	*n*	Range	Mean (s.d.)	*n*	Range	Mean (s.d.)
Strategy feedback	SC-IA strategy		3	0-1	0.06 (0.2)	22	0-4	0.69 (1.02)	8	0-1	0.15 (0.36)
	SA-IA strategy		0	0	0(0)	0	0	0(0)	0	0	0(0)
	IA-SC/SA strategy		18	0-15	1.37 (3.11)	19	0-14	1.81 (3.27)	4	0-3	0.13 (0.52)
	Test-in-parts strategy		0	0	0(0)	42	0-9	2.23 (2.13)	23	0-6	0.83 (1.26)
	Model-Build strategy		32	0-8	1.79 (2.17)	35	0-10	2.04 (2.43)	30	0-10	1.33 (1.82)
	Total strategy feedback		38	0-20	3.2 (3.9)	50	0-18	6.77 (5.05)	39	0-16	2.44 (2.9)
Task oriented feedback	Conceptual model building	Conceptual entities	4	0-8	0.29 (1.26)	43	0-22	4.77 (4.31)	31	0-13	2.5 (3.3)
		Conceptual set of behaviors	0	0	0(0)	7	0-5	0.29 (0.99)	1	0-1	0.02 (0.1)
		Sense-act framework	52	2-37	11.86 (8.0)	22	0-12	1.98 (3.05)	21	0-7	1.17 (1.7)
		Total conceptual modeling feedback	52	2-37	12.15 (7.94)	49	0-24	7.04 (5.85)	46	0-14	3.69 (3.3)
	Computational model building		37	0-11	2.23 (2.56)	42	0-16	3.46 (4.03)	10	0-5	0.35 (0.9)
	Total task based feedback		52	2-45	14.38 (8.69)	50	0-37	10.5 (8.52)	46	0-16	4.04 (3.5)

We found that students needed a combination of task and strategy feedback in all the modeling activities. In the initial rollercoaster activity, students received more task oriented feedback than in the other two activities. In the more complex fish-macro activity with multiple agents and behaviors, students needed more strategy feedback than in the rollercoaster activity, but less task oriented feedback than in the rollercoaster activity, implying that the effects of the feedback persisted across units. However, students found it challenging to manage and integrate the different tasks in a complex modeling activity involving a new domain. Finally, in the fish-micro activity, the task feedback received was further reduced, and the strategy feedback also decreased (to a smaller number than in the initial rollercoaster activity). This result provides preliminary evidence that our scaffolding effects persisted, and, therefore, a fading effect occurred naturally as students worked across units. Further, the resulting conceptual and computational models in the fish-micro activity were the most accurate of any activity, even though the students received less feedback in each category of scaffolds than in the earlier activities.

For the task oriented feedback, we noticed that students needed a combination of conceptual and computational model building feedback in all the activities. Looking specifically at the conceptual modeling scaffolds, we find that almost all of the feedback in the rollercoaster activity was directed at correctly conceptualizing sense-act processes. However, students got significantly better in conceptualizing sense-act processes in the fish macro and fish micro activities. In these activities, most of the conceptual model building scaffolds were directed at correctly conceptualizing the right set of entities in the domain, which may be attributed to students' low prior knowledge in the ecology domain (see pre-test scores in Table 7). Students were faced with learning and modeling new domain content with multiple agents and environment elements in the fish macro and fish micro activities.

With respect to strategy feedback, we see that students needed a combination of the different scaffolds except that for the *SA-IA* strategy. The value 0 across all activities for the *SA-IA* feedback is unusual, but that was because the condition under which this strategy was triggered was rarely assessed. This result implies that this assessment needs to be further refined in the learner model in the future. In general, students needed a lot of scaffolding for the *Model-Build* strategy, the *test-in-parts* strategy, which was applicable for the larger ecology activities, and the *IA-SC/SA* strategy. Again, these results show that the feedback on the five strategies and different aspects of the modeling tasks was effective, in that students learned how to use the strategies, and there was a general fading effect on the need for feedback across activities.

7.3 Discussion

In summary, this research study demonstrates the effectiveness of the CTSiM v2 environment with adaptive scaffolding, and provides valuable insight into the six research questions that this study was designed to investigate. We compare students who receive and who do not receive adaptive scaffolding and draw the following conclusions about the impact of the adaptive scaffolding:

i. Students who receive the adaptive scaffolding show higher science and CT learning gains compared to students who do not receive the adaptive scaffolding.
ii. Students who are provided adaptive scaffolding build more accurate conceptual and computational models, and are better able to transfer their modeling skills to new scenarios, in comparison to students who are not provided with adaptive scaffolding
iii. Students who are adaptively scaffolded display more proficient modeling behaviors, better use of CT practices, and a higher frequency of effective strategy use, compared to students who are not adaptively scaffolded

Moreover, our results show that students' modeling performances and modeling behaviors in the CTSiM v2 environment are correlated with their science learning. In particular, we find strong correlations between science learning and the use of important CT practices, like decomposing modeling tasks and understanding and relating representations at different levels of abstraction. This correlation clearly demonstrates the synergy between CT and science learning, which we have tried to leverage in the CTSiM environment to foster CT skills and science learning simultaneously.

Besides comparing students who received and did not receive scaffolding, we also analyzed how the type and frequency of scaffolds needed by students varied across time. We have shown that students needed a combination of task-based and strategy-based scaffolds in all activities, and that the average number of scaffolds required of each type decreased with time across activities. This result, combined with students' modeling proficiency in the final activity and their high learning gains, demonstrates the fading effect of our scaffolds.

CHAPTER 8

Discussion, Conclusions, and Future Work

Computational Thinking (CT) represents a fundamental set of skills that can be linked to problem formulation and solving, such as representing problems at different levels of abstraction, decomposing complex problems into manageable parts, and reformulating problems in terms of ones with known solutions. CT engages the power of computing to provide the framework for representing, analyzing, and solving problems in diverse disciplines. Of particular interest to us are the links between CT and STEM learning. CT is considered to be at the core of all STEM disciplines, and they share several common epistemic and representational practices. While computational mechanisms are better learned when contextualized in real-world problem contexts including those related to STEM domains, STEM concepts and fundamental science laws also become easier to understand and apply using computational representations and mechanisms. With the current emphasis on promoting both CT skills and STEM awareness for developing a globally competitive 21st century workforce, finding ways to leverage the synergy between CT and STEM becomes critical.

In today's world, it is imperative that students learn CT skills and be able to apply them starting at an early age. While the importance of introducing CT in the K-12 curricula has been emphasized by several researchers, the field of CT research has concentrated a lot of its effort into motivating and encouraging students to pursue computer science and use of computational tools through extra-curricular game-design or app-design activities. Such activities generally do not try to connect their activities and learning goals to existing K-12 standards or STEM learning concepts. Broadening participation in CS through motivational extracurricular CT-based activities may be a good first step, but CT eventually needs to be integrated into the K-12 curricula to make it accessible to everybody, including minorities and women. Also, curricular integration of CT requires development of systematic assessments for evaluating students and scaffolds for helping students with their difficulties, areas which both require further research and development.

This dissertation research makes significant contributions to the field of CT in K-12 education by demonstrating how to successfully leverage the synergy between CT and middle school science learning. We have developed CTSiM – a computer-based learning environment that middle school students use to build computational models of science topics. Running a study with an initial version of the CTSiM, we found that computational modeling of science topics helped students learn the relevant science content, but students faced a number of challenges and required continual individualized scaffolding for overcoming the challenges. By analyzing students' challenges, we were able to design and implement a number of modifications to the CTSiM system, and add adaptive scaffolds to help students become better learners and problem solvers. Our results show that the science learning gains for the experimental group that received adaptive scaffolding while working in the CTSiM v2 environment were higher than that of students who received one-on-one individualized scaffolding from members of our research team when working with the initial CTSiM system (the Effect size for the kinematics learning gains was 0.88 for CTSiM v2 versus 0.71 for the initial system, and the Effect size for ecology learning gains increased to 3.25 in CTSiM v2 from 3.16 in CTSiM v1)[1]. This shows that the modifications made to the CTSiM v2 environment to facilitate model building and model verification, along with the adaptive scaffolds helped achieve similar, or better learning gains than that achieved with the initial version of CTSiM where students received dedicated and individualized assistance from researchers to overcome their challenges. Also, the science learning gains of students who used the CTSiM v2 environment without adaptive scaffolding were higher than that of students who used CTSiM v1 without individualized assistance (the Effect size for the kinematics learning gains was 0.55 in CTSiM v2 as opposed to 0.05 in the initial version; the Effect size for ecology learning gains was 1.35 for CTSiM v2 versus 1.09 for CTSiM v1). From this, we concluded that the additional scaffolding and tools we provided through system redesign (see Chapter 5) had a positive effect on student learning.

[1] (The comparison of the effect sizes across the two studies is presented merely as a qualitative measure, since the research methodologies were not identical in the two studies, and the test questions in the second study were a refined version of those used in the first study. Also, we did not compare students in the two studies using measures external to the CTSiM environment.

Overall, the primary contributions of this dissertation can be summarized as follows:

i. Designing and Developing CTSiM - a computer-based learning environment that adopts the learning by modeling paradigm, and combines agent-based modeling paradigm with visual programming to simultaneously fosters CT skills and middle school science learning,
ii. Designing and Developing adaptive scaffolds for CT-based science learning - the need for scaffolding is determined using a combination of the students' modeling behaviors and their modeling performance, and
iii. Developing multiple modes of assessments for CT-based science learning and evaluating the effectiveness of the adaptive scaffolds using these assessments.

8.1 Contributions to the development of CT-based learning environments which can be integrated into middle school science classrooms

Unlike several CT-based environments and activities that are anchored in contexts like game-design, storytelling, and app-design, the CTSiM learning environment requires students to construct simulation models of science topics by carefully combining information acquisition, solution construction, and solution assessment tasks. The CTSiM design provides an example of how CT principles can be operationalized and successfully integrated with existing science curricula to help students simultaneously learn

(i) science concepts in kinematics and ecology,
(ii) computational concepts like conditional logic, use of sequences, loops, operators, and variables, and
(iii) important CT practices such as algorithmic thinking, abstraction, modularization, decomposition, incremental and iterative problem solving, and model verification using testing and debugging.

For example, the linked conceptual and computational modeling representations support decomposition and using abstractions to deal with complex modeling tasks. Similarly, the feature that supports checking and comparing subsets of modeled behaviors highlights CT practices like testing in parts.

Further, the intuitive agent-based, visual programming paradigm, and the domain specific modeling language support seamless integration into existing middle school science classrooms. CTSiM has been used successfully by a number of middle school science teachers in different cities in U.S.A., with and without researchers from our team being present when it was used in the classroom. All the teachers reported that they had no prior programming experience, but found computational modeling using CTSiM intuitive, and the 'Programming Guide' resources included in CTSiM extremely helpful for introducing their students to agent-based conceptual and computational modeling, and the use of common computational constructs for building models of scientific processes. Teacher reports and our observations in classrooms also confirm that students generally find modeling using CTSiM intuitive, enjoyable, and engaging.

Not only do students find working in the CTSiM environment enjoyable, they also demonstrate strong learning gains in both the science and CT domains. While our results in Chapter 7 illustrate that adaptive scaffolding can improve learning gains, we notice that even students who do not receive adaptive scaffolding while working with CTSiM show significant ($p < 0.0001$) pre-to-post learning gains for CT content as well as science content in Kinematics and Ecology. In addition, we also found that all students' science learning gains were correlated with their CT learning gains ($r = 0.2, p < 0.05$). Further, students' use of desired CT practices were also correlated with their science learning, especially in the Ecology units (see Table 14 in Section 7.2.5), again establishing the synergy between science and CT learning using CTSiM.

8.2 Contributions to the development of adaptive scaffolds for CT-based science learning environments

Model building in CTSiM v2 is a complex task. The environment provides a number of supporting tools, such as hypertext based searchable domain and CT resources, linked conceptual and computational modeling representations that help students decompose the complex model and build it in parts, a block-structured visual language to build the computational models, the ability to step through blocks to test the evolving simulation function, and a compare function that lets students compare the behaviors generated by their model against the behaviors generated by an expert model in parts. However, novice students find it difficult to combine all of the tools and scaffolds provided in the environment in an effective manner. Thus, we have developed a task-

and strategy-oriented learner modeling scheme that tracks and interprets students' actions in the CTSiM v2 environment by combining information about students' modeling behaviors and performances using 'coherence' and 'effectiveness' measures. The learner model then forms the basis for providing students with adaptive task and strategy based feedback contextualized in science domain content using a mixed initiative conversational dialog framework. We have demonstrated the effectiveness of our approach through a study run with control (no adaptive scaffolds) and experimental (with adaptive scaffolds) conditions. The experimental group outperformed the control group in (i) domain and CT learning gains, (ii) constructing correct science models, (iii) integrating the provided modeling representations, (iv) transferring modeling skills to a new scenario, and (v) use of the set of desired strategies we tracked in the system. Further, we noticed that students in the experimental condition required less task and strategy scaffolds across activities from the rollercoaster modeling activity to the fish-micro activity. The fact that the scaffolds can be 'faded' further substantiates their effectiveness.

Overall, our approach to learner modeling and scaffolding differs from the work of other researchers in a number of ways. Other learning-by-modeling environments for science domains generally scaffold students by (i) providing them with an assessment of their science models through model-driven simulations, or (ii) using learner models to give feedback on specific incorrect relationships modeled by students or specific modeling actions not taken. With respect to CT-based learning environments, very few provide any adaptive scaffolding at all. Environments like AgentSheets, which provide scaffolds are an exception, but even they merely provide automated assessments of students' computational artifacts by comparing CT patterns in students' artifacts with expected CT patterns.

Also, our task- and strategy-based learner modeling and adaptive scaffolding framework is not specific to CTSiM, but is generalizable for any OELE. Some researchers like Gobert et al. (2013) have claimed that using pre-determined metrics to assess learner actions in an OELE is problematic, and, to overcome this, they have applied educational data mining techniques to develop assessment metrics to evaluate student work. While it is true that students may use a variety of strategies to select and apply skills, and engineering metrics that take into account all potential corner cases is difficult, our end-goal in CTSiM is not merely developing an accurate assessment metric for students' task performance or strategy use. Rather, the assessment information that forms the basis of our learner models, is used primarily to provide feedback to students online and in the context of their current task, but only when it is clear that their performance metrics (i.e., effectiveness) is below pre-specified thresholds. Hence, our focus in this dissertation has not been on developing a comprehensive list of rules for specifying effective and ineffective task performance and strategy use. Our approach concedes that there can be various strategies and multiple ineffective variants of a strategy, but we chose ineffective strategy variants to detect and scaffold based on our observations in previous studies conducted with CTSiM v1. In the future, we also plan to use offline sequence mining techniques to derive common behavior patterns, and then use the patterns to track student behavior in future versions of CTSiM.

8.3 Contributions to the development of assessments for CT-based learning environments

Another contribution of this dissertation is the development of various assessments artifacts and metrics for the CTSiM learning environment. While we have used these assessments in the CTSiM environment to evaluate the effectiveness of our adaptive scaffolding framework, many of them can be generalized to CT-based learning environments in general. For example, our paper-based CT pre/post tests can be used as assessments for any CT-based environments where the target CT concepts include algorithmic thinking, conditional logic, use of loops, operators and variables, and understanding flow of control in a program. For CT–based environments that emphasize additional CT concepts like parallel and recursive thinking, efficiency and performance constraints, and data visualizations and analysis, additional assessment questions will be required. Similarly, our task- and strategy-based learner modeling and adaptive scaffolding framework using ‘effectiveness’ and ‘coherence’ relations can be applied to any OELE, and the modeling behavior metrics for frequency of use of desired and suboptimal strategies can be generalized to other OELEs including CT-based learning environments. While our model performance metrics are influenced by the specifics of the domain-specific modeling language used to build CTSiM models, our model evolution metrics can be used to assess how students build their computational artifacts in any CT-based environment as long as the correct or expected artifact is well defined.

Overall, the assessments we have developed for the CTSiM environment have been able to demonstrate the effectiveness of this dissertation research. We report strong and synergistic learning gains for science and CT, and show how the CTSiM modeling interfaces naturally promote important CT practices, which are also synergistic with science learning. The linked conceptual and computational modeling interfaces in CTSiM promote modeling at different levels of abstraction and decomposing the modeling task into modeling individual agent behaviors separately. The interfaces for observing whole or partial student models as simulations, on the other hand, promote systematic testing, debugging, and iterative refinement practices. In addition, we demonstrate that interpreting students' actions in CTSiM by combining information about their modeling behaviors and performance, and adaptively providing context-relevant conversational scaffolds helps students with their model building process. Adaptive scaffolding provided by a computer-based pedagogical mentor agent helps students build more accurate science models using the CTSiM environment, and combine information from their information acquisition, model construction, and model assessment tasks more effectively, which in turn results in higher science and CT learning gains.

8.4 Future research directions

This dissertation research represents a starting point for developing more advanced CT-based science learning environments with adaptive scaffolding that can be integrated into middle school classrooms. Additional research is needed in order to develop a more encompassing set of adaptive scaffolding strategies by making more productive use of the information derived about students' learning behaviors. In addition, lessons learned from the experiments conducted with the current version of CTSiM provides us with a framework to develop curricular units in other science domains. It is also important that we extend the use of CTSiM to more diverse populations, especially in schools that have a predominantly minority student population. We summarize some of these promising research directions below.

Refining the learner modeling and adaptive scaffolding strategy - In future versions of CTSiM, we plan to develop and maintain a more nuanced learner model which captures global information about students' modeling performance, and effective and ineffective uses of a more comprehensive list of strategies. We plan to define new strategies by looking at action sequences from our task model, and also using offline sequence mining techniques to identify common action patterns. Also, based on the nuanced learner model, we plan to refine our adaptive scaffolding strategy. Instead of providing scaffolds based on a simplified frequency count of ineffective strategies, we plan to trigger scaffolding feedback based on students' global behavior and distribution of effective and ineffective strategies. In addition, we plan to analyze students' action logs further to study students' responses to individual feedback instances and how well they were able to engage with the feedback and apply it to their model building and problem solving tasks. This will help us understand which forms of feedback students considered most useful, and how best to provide such feedback prompts and hints in the context of the students' current tasks.

Developing scaffolds for assisting teachers – Currently, we use information about students' modeling performances and behaviors to adaptively scaffold students. In classroom settings, it would also be helpful to provide teachers with assessments of how their class is doing, common challenges being faced, and point out specific students who seem to require individualized assistance. Developing a teacher dashboard with aggregate class data as well as information about individual students can scaffold teachers and assist them in managing classroom instruction in a more effective manner. The teacher can discuss common mistakes and problems with the whole class, and individually help students who are performing below the class average.

Emphasizing new CT concepts and practices – Currently, CTSiM learning activities focus on several important CT concepts and practices like algorithmic flow of control, conditional and iterative logic, use of variables, modularization and decomposition, abstraction, and testing and refining. However, we could also incorporate some other essential CT skills into CTSiM like data collection, visualization and analysis of collected data or data generated by simulating the models, systematic debugging of computational modules, and reusing and remixing of code. Data collection could involve real-world experiments or even running simulations in the CTSiM environment with different initial conditions. Providing an interface for students to systematically record their collected data, visualize it using multiple representations and understand the affordances of different representations, and draw conclusions and answer questions based on the collected data, could help foster some essential CT skills. Debugging is another important CT skill which is now implicit in the CTSiM environment, but we could promote it explicitly by developing an interface to help students with systematic debugging, testing in parts, and recording experiments run, conclusions drawn, and corresponding plans of action. Also, recognizing and understanding commonalities between program segments and being able to reuse old code by changing parameters, is an essential CT skill we could try and promote in the CTSiM environment. If we allow students to copy and paste blocks or import blocks from one procedure to another in an activity, or between activities, we can promote this skill and track students' actions to further provide feedback on how to reuse code efficiently.

Using CTSiM to teach more diverse science topics – Developing new learning activities for CTSiM which align with middle school science curricular standards will help demonstrate the generalizability of the CTSiM design and make the learning environment more useful for a wider population of teachers. Also, a number of CTSiM activities could entail a longer learning activity progression interspersed with non-CTSiM classroom activities. Exposing students to computational modeling and CT practices over a period of time across different science topics can help students develop a deeper understanding of computational methods and practices. In order to make developing new CTSiM activities easy, we plan to build authoring tools for the same. This would also allow teachers to design and develop their own learning activities.

Testing whether the effectiveness of CTSiM is generalizable to diverse student and teacher populations – We have currently evaluated CTSiM in a limited number of middle school classrooms with a small number of science teachers. We plan to expand our scope by testing CTSiM in many more middle schools spanning more diverse student and teacher populations, since the eventual goal is to make CT accessible to all students irrespective of their socio-economic demographics and academic proficiencies. A large scale testing of CTSiM also requires more attention to developing and validating CT assessments – a vital research area we plan to continue working on.

REFERENCES

Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal*, *55*(7), 832-835.

Ainsworth, S. (2006). DeFT: A conceptual framework for considering learning with multiple representations. *Learning and Instruction*, *16*(3), 183-198.

Allan, W. A., Erickson, J.L., Brookhouse, P., & Johnson, J.L. (2010). Development Through a Collaborative Curriculum Project – an Example of TPACK in Maine. *TechTrends , 54(6)*

Assaf, D., Escherle, N., Basawapatna, A., Maiello, C., & Repenning, A. (2016). Retention of Flow: Evaluating a Computer Science Education Week Activity. To appear in SiGCSE 2016, Memphis.

Azevedo, R. (2005). Using hypermedia as a metacognitive tool for enhancing student learning? The role of self-regulated learning. Educational Psychologist, 40(4):199–209.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community? *ACM Inroads*, *2*(1), 48-54.

Bar-Yam, Y. (1997). *Dynamics of complex systems* (Vol. 213). Reading, MA: Addison-Wesley.

Basawapatna, A. R., Repenning, A., & Koh, K. H. (2015, February). Closing The Cyberlearning Loop: Enabling Teachers To Formatively Assess Student Programming Projects. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 12-17). ACM.

Basu, S., Kinnebrew, J., Dickes, A., Farris, A.V., Sengupta, P., Winger, J., & Biswas, G. (2012). A Science Learning Environment using a Computational Thinking Approach. In *Proceedings of the 20th International Conference on Computers in Education* (pp. 722-729). Singapore.

Basu, S., Dickes, A., Kinnebrew, J.S., Sengupta, P., & Biswas, G. (2013). CTSiM: A Computational Thinking Environment for Learning Science through Simulation and Modeling. In *Proceedings of the 5th International Conference on Computer Supported Education* (pp. 369-378). Aachen, Germany.

Basu, S., Kinnebrew, J., & Biswas, G. (2014). Assessing student performance in a computational-thinking based science learning environment. In *Proceedings of the 12th International Conference on Intelligent Tutoring Systems* (pp. 476-481). Honolulu, HI, USA.

Basu, S., Dukeman, A., Kinnebrew, J., Biswas, G., & Sengupta, P. (2014). Investigating student generated computational models of science. In *Proceedings of the 11th International Conference of the Learning Sciences* (pp. 1097-1101). Boulder, CO, USA.

Basu, S., Sengupta, P., & Biswas, G. (2015). A scaffolding framework to support learning of emergent phenomena using multi-agent based simulation environments. *Research in Science Education, 45*(2), 293-324. DOI: 10.1007/s11165-014-9424-z

Basu, S., Biswas, G., Kinnebrew, J., & Rafi, T. (2015). Relations between modeling behavior and learning in a Computational Thinking based science learning environment. In *Proceedings of the 23rd International Conference on Computers in Education* (pp. 184-189). China.

Basu, S., Biswas, G. & Kinnebrew, J.S. (2016). Using multiple representations to simultaneously learn computational thinking and middle school science. In *Thirtieth AAAI conference on Artificial Intelligence*. Phoenix, Arizona, USA.

Basu, S. & Biswas, G. (2016). Providing adaptive scaffolds and measuring their effectiveness in open ended learning environments. In *12th International Conference of the Learning Sciences*. Singapore.

Basu, S., Sengupta, P., Dickes, A., Biswas, G., Kinnebrew, J.S., & Clark, D. (in review). Identifying middle school students' challenges in computational thinking based science learning. *Research and Practices in Technology Enhanced Learning*.

Basu, S., Biswas, G., & Kinnebrew, J.S. (in review). Learner modeling for adaptive scaffolding in a computational thinking-based science learning environment. *User Modeling and User-Adapted interaction, Special Issue on Impact of Learner modeling*.

Blikstein, P., & Wilensky, U. (2009). An atom is known by the company it keeps: A constructionist learning environment for materials science using agent-based modeling. *International Journal of Computers for Mathematical Learning*, *14*(2), 81-119.

Bransford, J., & Schwartz, D. (1999). Rethinking transfer: A simple proposal with multiple implica-tions. *Review of Research in Education, 24*(1), 61-101.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada*.

Buechley, L., Peppler, K., Eisenberg, M., & Kafai, Y.B. (2013). Textile Messages: Dispatches from the World of Electronic Textiles and Education. New York: Peter Lang.

Burke, Q., & Kafai, Y. B. (2010, June). Programming & storytelling: opportunities for learning about coding & composition. In *Proceedings of the 9th International Conference on Interaction Design and Children* (pp. 348-351). ACM.

Chandler, P., & Sweller, J. (1992). The split-attention effect as a factor in the design of instruction. British Journal of Educational Psychology, 62(2), 233-246.

Chi, M. T. H., Slotta, J. D., & de Leeuw, N. (1994). From things to processes: A theory of conceptual change for learning science concepts. Learning and Instruction, 4, 27–43.

Chi, M. T. H. (2005). Common sense conceptions of emergent processes: why some misconceptions are robust. *Journal of the Learning Sciences, 14*, 161–199.

Chandler, P., & Sweller, J. (1992). The split-attention effect as a factor in the design of instruction. British Journal of Educational Psychology, 62(2), 233e246.

Computer Science Teachers Association, 2011. ACM ISBN: # 978-1-4503-0881-6

Conway, M. (1997). Alice: Easy to Learn 3D Scripting for Novices, Technical Report, School of Engineering and Applied Sciences, University of Virginia, Charlottesville, VA.

Cooper, S., Nam, Y. J., & Si, L. (2012, July). Initial results of using an intelligent tutoring system with Alice. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education* (pp. 138-143). ACM.

Danish, J. A., Peppler, K., Phelps, D., & Washington, D. (2011). Life in the hive: supporting inquiry into complexity within the zone of proximal development. *Journal of Science Education and Technology, 20*(5), 454–467.

Darwin, C. (1871). The descent of man, and selection in relation to sex (1st ed.), London: John Murray, ISBN 0801420857, retrieved 2009-06-18

Dasgupta, S., Clements, S. M., Idlbi, A. Y., Willis-Ford, C., & Resnick, M. (2015, October). Extending Scratch: New pathways into programming. In*Visual Languages and Human-Centric Computing (VL/HCC), 2015 IEEE Symposium on* (pp. 165-169). IEEE.

Dickes, A., & Sengupta, P. (2013). Learning natural selection in 4th grade with multi-agent-based computational models. *Research in Science Education, 43*(3), 921–953.

Dillenbourg, P. (1999). Collaborative learning: cognitive and computational approaches. Advances in Learning and Instruction Series. Elsevier Science, Inc., PO Box 945, Madison Square Station, New York, NY 10160–0757.

DiSalvo, B., & Bruckman, A. (2011). From interests to values. *Communications of the ACM*, *54*(8), 27-29.

diSessa, A. A., Abelson, H., & Ploger, D. (1991a). An overview of boxer. Journal of Mathematical Behavior, 10(1), 3–15.

diSessa, A., Hammer, D., Sherin, B., & Kolpakowski, T. (1991b). Inventing graphing: Children's metarepresentational expertise. Journal of Mathematical Behavior, 10(2), 117–160.

diSessa, A. A. (2000). *Changing minds: Computers, learning, and literacy*. Cambridge, MA: MIT Press.

Duque, R., Bollen, L., Anjewierden, A., & Bravo, C. (2012). Automating the Analysis of Problem-solving Activities in Learning Environments: the Co-Lab Case Study. J. UCS, 18(10), 1279-1307.

Duschl, R. A., Schweingruber, H. A. & Shouse, A. W. (2007). *Taking Science to School: Learning and Teaching Science in Grades K-8*. National Academies Press.

Dykstra, D. I., Jr., & Sweet, D. R. (2009). Conceptual development about motion and force in elementary and middle school students. American Journal of Physics, 77(5), 468–476.

Elby, A. (2000). What students' learning of representations tells us about constructivism. Journal of Mathematical Behavior, 19, 481–502.

Elsom-Cook, M. (1993). Student modelling in intelligent tutoring systems. *Artificial Intelligence Re-view*, *7*(3-4), 227-240.

Forte, A., & Guzdial, M. (2004). Computers for communication, not calculation: Media as a motivation and context for learning. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on* (pp. 10-pp). IEEE.

Goldman, S. R. (2003). Learning in complex domains: When and why do multiple representations help? *Learning and instruction*, *13*(2), 239-244.

Goldstone, R. L., & Wilensky, U. (2008). Promoting transfer by grounding complex systems principles. *Journal of the Learning Sciences, 17*(4), 465–516.

Grover, S., & Pea, R. (2013). Computational Thinking in K–12 : A Review of the State of the Field. *Educational Researcher*, 42(1), 38-43

Grover, S., Cooper, S., & Pea, R. (2014, June). Assessing computational learning in K-12. In *Proceedings of the 2014 conference on Innovation & technology in computer science education* (pp. 57-62). ACM.

Grover, S., Pea, R., & Cooper, S. (2014). Expansive framing and preparation for future learning in middle-school computer science. In *International Conference of the Learning Sciences (ICLS) Conference* (pp. 992-996).

Guzdial, M. (1994). Software-realized scaffolding to facilitate programming for science learning. *Interactive Learning Environments*, *4*(1), 001-044.

Guzdial, M. (1995, March). Centralized mindset: a student problem with object-oriented programming. In *ACM SIGCSE Bulletin* (Vol. 27, No. 1, pp. 182-185). ACM.

Guzdial, M. (2010). Does contextualized computing education help?. *ACM Inroads*, *1*(4), 4-6.

Halloun, I. A., & Hestenes, D. (1985). The initial knowledge state of college physics students. American Journal of Physics, 53(11), 1043–1056.

Hambrusch, S., Hoffmann, C., Korb, J. T., Haugan, M., & Hosking, A. L. (2009). A multidisciplinary approach towards computational thinking for science majors. *ACM SIGCSE Bulletin*, *41*(1), 183-187.

Hemmendinger, D. (2010). A plea for modesty. *Acm Inroads*, *1*(2), 4-7.

Hmelo-Silver, C. E., & Pfeffer, M. G. (2004). Comparing expert and novice understanding of a complex system from the perspective of structures, behaviors, and functions. *Cognitive Science, 28*(1), 127–138.

Hohmann, L., Guzdial, M., & Soloway, E. 1992. SODA: A computer-aided design environment for the doing and learning of software design. In Computer Assisted Learning (pp. 307-319). Springer Berlin Heidelberg

Holland, J. (1995). *Hidden order: how adaptation builds complexity*. Cambridge: Perseus.

Holland, J. (1998). *Emergence: from chaos to order*. Reading, MA: Perseus Books.

Hundhausen, C. D., & Brown, J. L. (2007). What You See Is What You Code: A "live" algorithm development and visualization environment for novice learners. Journal of Visual Languages and Computing, 18, 22–47.

International Society for Technology in Education (ISTE), National Educational Technology Standards for Students (NETS), 2nd ed., 2007.

Jacobson, M. J. (2001). Problem solving, cognition, and complex systems: differences between experts and novices. *Complexity, 6*(3), 41–49.

Jacobson, M. J., & Wilensky, U. (2006). Complex systems in education: scientific and education importance and implications for the learning sciences. *The Journal of the Learning Sciences, 15*(1), 11–34.

Jona K, Wilensky U, Trouille L, Horn MS, Orton K, Weintrop D, Beheshti E. (2014). Embedding computational thinking in science, technology, engineering, and math (CT-STEM). In: Paper presented at the future directions in computer science education summit meeting, Orlando, FL

Kafai, Y. B. (1995). *Minds in Play*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Kafai, Y. B., Searle, K., Kaplan, E., Fields, D., Lee, E., & Lui, D. (2013, March). Cupcake cushions, scooby doo shirts, and soft boomboxes: e-textiles in high school to promote computational concepts, practices, and perceptions. In *Proceeding of the 44th ACM technical symposium on Computer science education* (pp. 311-316). ACM.

Kahn, K. (1996). ToonTalk: An animated programming environment for children. Journal of Visual Languages and Computing.

Kapur, M., & Kinzer, C. K. (2009). Productive failure in CSCL groups. *International Journal of Computer-Supported Learning, 4*(1), 21–46.

Kauffman, S. (1995). *At home in the universe: the search for laws of self-organization and complexity*. New York: Oxford University Press.

Kelleher, C. & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys*, 37, 2, 83-137.

Kelleher, C., & Pausch, R. (2007). Using storytelling to motivate programming. *Communications of the ACM*, *50*(7), 58-64.

Kelleher, C. (2008). Learning computer programming as storytelling. *Beyond Barbie and Mortal Kombat: New perspectives on gender and computer games*, 247-264.

Kinnebrew, J.S., Loretz, K.M., & Biswas, G. (2013). A Contextualized, Differential Sequence Mining Method to Derive Students' Learning Behavior Patterns. Journal of Educational Data Mining, 5(1), 190-219.

Kinnebrew, J.S., Segedy, J.R., & Biswas, G. (2014). Analyzing the Temporal Evolution of Students' Behaviors in Open-Ended Learning Environments. Metacognition and Learning, 9(2), 187-215.

Kinnebrew, J., Segedy, J.R. & Biswas, G. (2016). Integrating Model-Driven and Data-Driven Techniques for Analyzing Learning Behaviors in Open-Ended Learning Environments. IEEE Transactions on Learning Technologies. *In Press

Klopfer, E. (2003). Technologies to support the creation of complex systems models—using StarLogo software with students. *Biosystems, 71*(1), 111–122.

Klopfer, E., Scheintaub, H., Huang, W, Wendel, D. (2009). StarLogo TNG: Making Agent Based Modeling Accessible and Appealing to Novices In Artificial Life Models in Software.

Koedinger, K. R., & Aleven, V. (2007). Exploring the assistance dilemma in experiments with cogni-tive tutors. Educational Psychology Review, 19(3), 239-264.

Koh, K.H., Basawapatna, A., Bennett, V., Repenning, A. Towards the Automatic Recognition of Computational Thinking for Adaptive Visual Language Learning. (2010) VL/HCC: 59-66

Kolodner, J. L., Camp, P. J., Crismond, D., Fasse, B., Gray, J., Holbrook, J., & Ryan, M. (2003). Problem-based learning meets case-based reasoning in the middle-school science classroom: Putting learning by design (tm) into practice. *The journal of the learning sciences*, *12*(4), 495-547.

Kynigos, C. (2007). Using half-baked microworlds to challenge teacher educators' knowing. *International journal of computers for mathematical learning*, *12*(2), 87-111.

Land, S. (2000). Cognitive requirements for learning with open-ended learning environments. *Educational Technology Research and Development, 48*(3), 61-78.

Land, S., Hannafin, M., & Oliver, K. (2012). Student-centered learning environments: Foundations, assumptions and design. In D. Jonassen & S. Land (Eds.), *Theoretical Foundations of Learning Environ-ments* (pp. 3-25). New York, NY: Routledge.

Larkin, J. H., McDermott, J., Simon, D. P., & Simon, H. A. (1980). Expert and novice performance in solving physics problems. Science, 208, 1335–1342.

Lehrer, R., & Schauble, L. (2006). *Cultivating model-based reasoning in science education*.

Leinhardt, G., Zaslavsky, O., & Stein, M. M. (1990). Functions, graphs, and graphing: Tasks, learning and teaching. Review of Educational Research, 60, 1–64.

Levy, S. T., & Wilensky, U. (2008). Inventing a "mid-level" to make ends meet: Reasoning through the levels of complexity. Cognition and Instruction, 26(1), 1–47.

Luckin, R. and du Boulay, B. (1999). Ecolab: The development and evaluation of a Vygotskian design framework. International Journal of Artificial Intelligence in Education, 10(2):198–220. Mitrovic, A.: 2011, 'Fifteen years of Constraint-Based Tutors: What we have achieved and where we are going'. User Modeling and User-Adapted Interaction.

Macal, C. M., & North, M. J. (2008). Agent-based modeling and simulation: ABMS examples. In Proceedings of the 40th Conference on Winter Simulation (pp. 101-112). Winter Simulation Conference.

Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., & Resnick, M. (2004) Scratch: A sneak preview. In Proceedings of Creating, Connecting, and Collaborating through Computing, 104–109.

Martin, T., Berland, M., BenTon, T., & SMiTh, C. P. (2013). Learning programming with IPRO: The effects of a mobile, social programming environment. *Journal of Interactive Learning Research*, *24*(3), 301-328.

Mataric, M. J. (1993). Designing emergent behaviors: from local interactions to collective intelligence. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior* (pp. 432–441), August.

Maxwell, J.C. (1871). Remarks on the mathematical classification of physical quantities. *Proceedings of the London Mathematical Society, s1–3*, 224–233.

Mayer, R. E., & Moreno, R. (2002). Aids to computer-based multimedia learning. Learning and Instruction, 12(1), 107-119.

McCloskey, M. (1983). Naive theories of motion. In D. Gentner & A. Stevens (Eds.), Mental models (pp. 299–324). Hillsdale, NJ: Lawrence Erlbaum.

Moskal, B., Lurie, D., & Cooper, S. (2004, March). Evaluating the effectiveness of a new instructional approach. In *ACM SIGCSE Bulletin* (Vol. 36, No. 1, pp. 75-79). ACM.

Mitchell, M. (2009). *Complexity: a guided tour.* New York: Oxford University Press.

National Research Council. (2008). Taking science to school: Learning and teaching science in grades K–8. Washington, DC: National Academy Press.

National Research Council. (2010). Committee for the Workshops on Computational Thinking: Report of a workshop on the scope and nature of computational thinking. Washington, DC: National Academies Press

National Research Council. (2011). *Committee for the Workshops on Computational Thinking: Report of a workshop of pedagogical aspects of computational thinking*. Washington, DC: National Academies Press.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York, NY: Basic Books.

Papert, S. (1991). Situating constructionism. In I. Harel & S. Papert (Eds.), *Constructionism*. Norwood, NJ: Ablex Publishing Corporation.

Parsons, D., & Haden, P. (2007). Programming osmosis: Knowledge transfer from imperative to visual programming environments. In *Procedings of The Twentieth Annual NACCQ Conference* (pp. 209-215).

Pathak, S. A., Jacobson, M. J., Kim, B., Zhang, B., & Feng D. (2008). *Learning the physics of electricity with agent based models: paradox of productive failure*. Paper presented at the International Conference in Computers in Education. Oct. 27–31 Taipei.

Pausch, R., Burnette, T., Capeheart, A.C., Conway, M., Cosgrove, D., DeLine, R., Durbin, J., Gossweiler, R., Koga, S., & White, J. (1995) Alice: Rapid Prototyping System for Virtual Reality. IEEE Computer Graphics and Applications.

Penner, D. E., Lehrer, R., & Schauble, L. (1998). From physical models to biomechanics: A design-based modeling approach. Journal of the Learning Sciences, 7(3–4), 429–449.

Piaget, J. (1929). The child's conception of the world. London: Routledge and Kegan Paul.

President's Information Technology Advisory Council (PITAC05), "Computational Science: Ensuring America's Competitiveness," Report to the President, June 2005.

Pressley, M., Goodchild, F., Fleet, J., Zajchowski, R., & Evansi, E. (1989). The challenges of class-room strategy instruction. *The Elementary School Journal, 89*, 301-342.

Puntambekar, S. and Hubscher, R. (2005). Tools for scaffolding students in a complex learning environment: What have we gained and what have we missed? Educational Psychologist, 40(1):1–12.

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*, 222–232.

Reiner, M., Slotta, J. D., Chi, M. T. H., & Resnick, L. B. (2000). Naive physics reasoning: A commitment to substance-based conceptions. Cognition and Instruction, 18(1), 1–34.

Repenning, A. (2000). AgentSheets®: An interactive simulation environment with end-user programmable agents. *Interaction*.

Repenning, A., Webb, D., Ioannidou, A. (2010). Scalable Game Design and the Development of a Checklist for Getting Computational Thinking into Public Schools, The 41st ACM Technical Symposium on Computer Science Education, SIGCSE, (Milwaukee, WI), ACM Press.

Resnick, M. (1996). Beyond the Centralized Mindset. Journal of the Learning Sciences, vol.5, no.1, pp.1-22.

Resnick, M. (2007). Sowing the Seeds for a More Creative Society. Learning and Leading with Technology.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., Kafai, Y. (2009). Scratch: Programming for All. Communications of the ACM, November 2009.

Schraw, G., Crippen, K. J., & Hartley, K. (2006). Promoting self-regulation in science education: Metacognition as part of a broader perspective on learning. *Research in Science Education*, *36*(1-2), 111-139.

Schwartz, D. L., & Arena, D. (2013). Measuring what matters most: Choice-based assessments for the digital age. MIT Press.

Segedy, J.R., Kinnebrew, J.S., & Biswas, G. (2013). The Effect of Contextualized Conversational Feedback in a Complex Open-Ended Learning Environment. Educational Technology Research and Development, 61(1), 71-89.

Segedy, J.R., Kinnebrew, J.S., and Biswas, G. (2015). Using coherence analysis to characterize self-regulated learning behaviours in open-ended learning environments. *Journal of Learning Analytics, 2*(1), 13–48.

Sengupta, P. (2011). Design Principles for a Visual Programming Language to Integrate Agent-based modeling in K-12 Science. In: Proceedings of the Eighth International Conference of Complex Systems (ICCS 2011), pp 1636–1637.

Sengupta, P., & Wilensky, U. (2009). Learning electricity with NIELS: thinking with electrons and thinking in levels. *International Journal of Computers for Mathematical Learning, 14*(1), 21–50.

Sengupta, P., & Wilensky, U. (2011). Lowering the learning threshold: Multi-agent-based models and learning electricity. In M. S. Khine & I. M. Saleh (Eds.), Dynamic modeling: Cognitive tool for scientific inquiry (pp. 141–171). New York, NY: Springer.

Sengupta, P., & Farris, A. V. (2012). Learning Kinematics in Elementary Grades Using Agent-based Computational Modeling: A Visual Programming Based Approach. *Proceedings of the 11th International Conference on Interaction Design & Children*, pp 78–87.

Sengupta, P., Kinnebrew, J. S., Basu, S., Biswas, G., & Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies*, *18*(2), 351-380.

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

Sherin, B. L. (2001). How students understand physics equations. *Cognition and instruction*, *19*(4), 479-541.

Sison, R., & Shimura, M. (1998). Student modeling and machine learning. *International Journal of Artificial Intelligence in Education (IJAIED)*, *9*, 128-158.

Smith, A. (1977). *An inquiry into the nature and causes of the wealth of nations*. University of Chicago Press.

Smith, D., Cypher, A., & Tesler, L. (2000). Programming by example: Novice programming comes of age. Communications of the ACM, 43(3), 75–81.

Soloway, E. (1993). Should we teach students to program?. *Communications of the ACM*, *36*(10), 21-24.

Soloway, E., Guzdial, M., & Hay, K. E. (1994). Learner-centered design: The challenge for HCI in the 21st century. *interactions*, *1*(2), 36-48.

Tan, J., & Biswas, G. (2007). Simulation-based game learning environments: building and sustaining a fish tank. In *Proceedings of the First IEEE International Workshop on Digital Game and Intelligent Toy Enhanced Learning* (pp. 73–80). Jhongli, Taiwan.

Tisue, S., & Wilensky, U. (2004). NetLogo: a simple environment for modeling complexity. In International Conference on Complex Systems (pp. 16–21), May.

UK Department for Education (UKEd13), "Computing Programmes of study for Key Stages 1-4," February 2013,http://media.education.gov.uk/assets/files/pdf/c/computing%2004-02-13_001.pdf

van der Meij, J., & de Jong, T. (2006). Supporting students' learning with multiple representations in a dynamic simulation-based learning environment. *Learning and Instruction*, *16*(3), 199-212.

Weber, G., & Specht, M. (1997, January). User modeling and adaptive navigation support in WWW-based tutoring systems. In *User Modeling* (pp. 289-300). Springer Vienna.

Werner, L., Campe, S., & Denner, J. (2012, February). Children learning computer science concepts via Alice game-programming. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (pp. 427-432). ACM.

White, B. Y., & Frederiksen, J. R. (1990). Causal model progressions as a foundation for intelligent learning environments. Artificial Intelligence, 42(1), 99–157.

Wieman, C.E., Adams, W.K., & Perkins, K.K. (2008). PhET research: simulations that enhance learning. *Science, 322*, 682-683.

Wilensky, U. (1999). *NetLogo.* Center for Connected Learning and Computer-Based Modeling (http://ccl.northwestern.edu/netlogo). Northwestern University, Evanston, IL.

Wilensky, U., & 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.

Wilensky, U., & Reisman, K. (2006). Thinking like a wolf, a sheep, or a firefly: learning biology through constructing and testing computational theories—an embodied modeling approach. *Cognition and Instruction, 24*(2), 171–209.

Wilensky, U., Brady, C. E., & Horn, M. S. (2014). Fostering computational literacy in science classrooms. *Communications of the ACM*, *57*(8), 24-28.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society, 366*, 3717–3725.

Wolber, D., Abelson, H., Spertus, E., & Looney, L. (2011). *App Inventor*. " O'Reilly Media, Inc.".

APPENDIX A

Information acquisition resources for CTSiM v2

A.1 Science resources

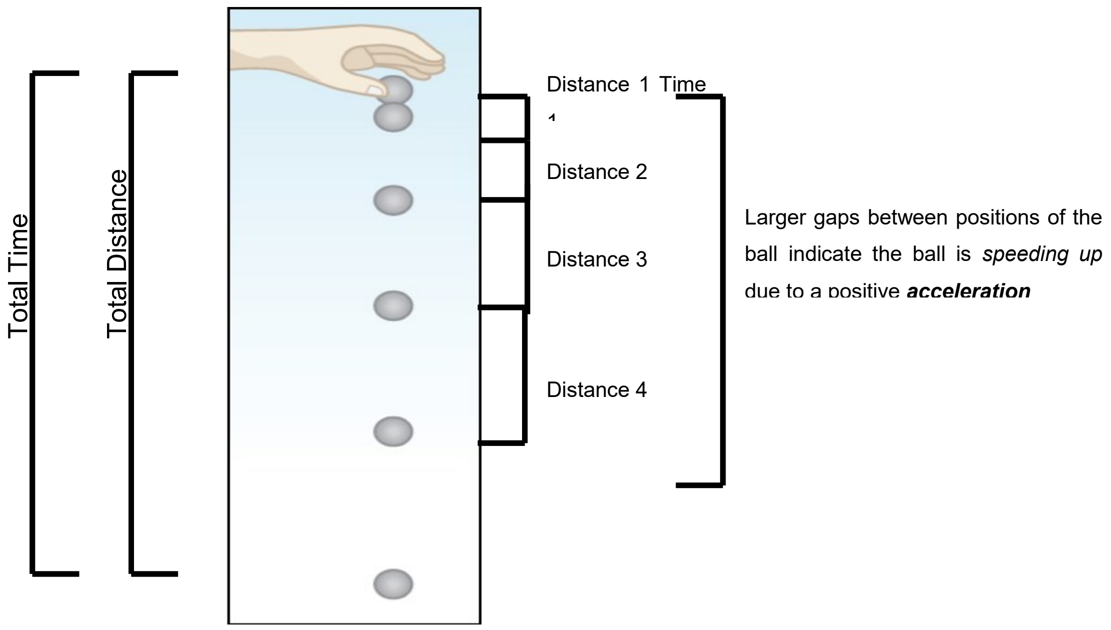
A.1.1 Kinematics unit

The hypertext resources used as part of the CTSiM v2 'Science Book' for the Kinematics unit are included on the following pages.

Motion

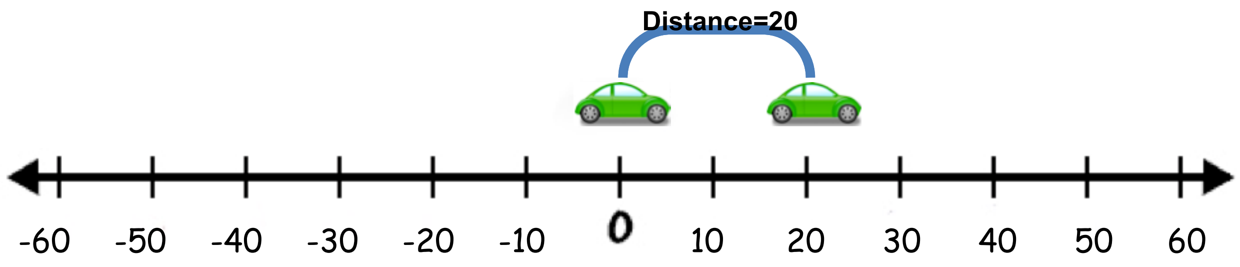
In physics, motion is a change in position of an object with respect to both time and the starting point of the object. In general terms, motion can be thought of as a process of continual change. In more specific terms, the motion of an object can de described in terms of the objects' position, speed and acceleration.

If we look at the figure below, we see a hand dropping a ball. The motion of this ball can be described in terms of its position, speed and acceleration. The total distance traveled is the distance between the starting point of the ball in the hand and the final recorded position (last picture of the ball). If we wanted to, we could measure the distance at different points in time, for instance between position 1 and position 2. This distance would be one part of the total distance traveled. Similarly, if we measure both the distance *between* positions of the ball and how long it takes to travel from one position to another, we can determine the speed of the ball. Speed describes how fast an object is moving. The speed of an object can remain constant with time or can increase or decrease with time. In the figure below, we can see that the speed of the ball at position 1, 2, 3 and 4 is different because the ball travels further in the same amount of time. This means that the speed is *not constant*. If the ball was traveling at a constant speed the size of the gaps for Speed 2, Speed 3 and Speed 4 would be of the same size. This speed up is due to acceleration caused by gravity. Please follow the links for position, speed and acceleration above to learn more about these concepts.


Total Time
Total Distance
Distance 1 Time 1
Distance 2
Distance 3
Distance 4
Larger gaps between positions of the ball indicate the ball is *speeding up* due to a positive ***acceleration***

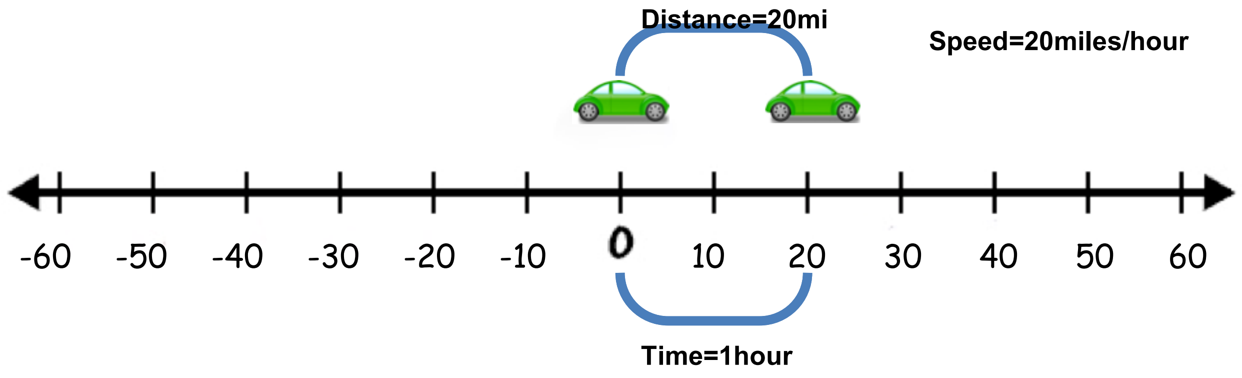
Position

The distance of an object is the total path length it has traveled from one position to another. For instance, if an object located along a number line started at position zero and traveled to position 20, the final position of that object would be 20 units of distance. In this example the unit of distance is undefined; however, if we had specified that the object had traveled from mile 0 to mile 20, the total distance traveled would be 20 miles. If you look at the figure below, you can see the car's starting position of zero and its final position of 20 illustrated.

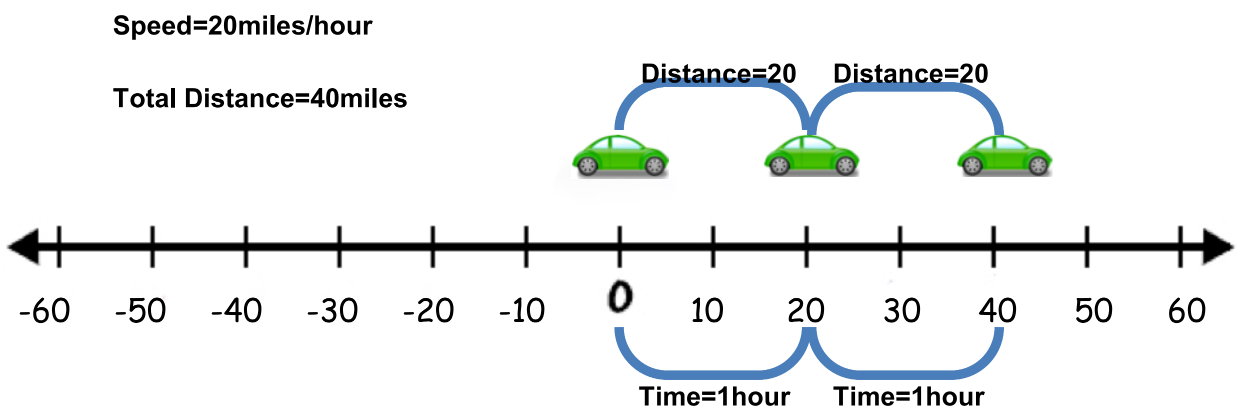


Speed

The speed of an object is related to its position. Speed is equal to the total distance traveled per unit time. In other words, speed is the rate of change of an object's position. For instance, if the object described above traveled 20 units of distance in 1 unit time, the speed of that object would be equal to 20 units of distance per unit time, or 20 distance units/time unit. As in the above example, units have been left undefined; however, if the object had traveled 20 miles in 1 second, the speed of that object would be 20miles/second. On the other hand, if the car had traveled 20 miles in 1 hour, its speed would be 20miles/hour



Speed can be either constant or non-constant. Constant speed indicates that the speed of an object does not change. This is similar to setting "cruise control" on a car. If you set a car's cruise control to 20miles/hour, the car will never travel faster or slower than 20miles/hour. If we know the constant speed of an object, we can easily predict the position of the object in the future. For instance, if we set a car's cruise control to 20miles/hour, we can predict that, starting from zero, the car will have traveled 40 miles in two hours and 80 miles in four hours. The figure below indicates constant speed as it is related to position and time. Notice that the spaces between positions of the car are the same length. Equal distance gaps indicate a constant speed.



Non-constant or variable speed indicates a changing speed. The amount of change, either or positive or negative, of the speed of an object is referred to as acceleration.

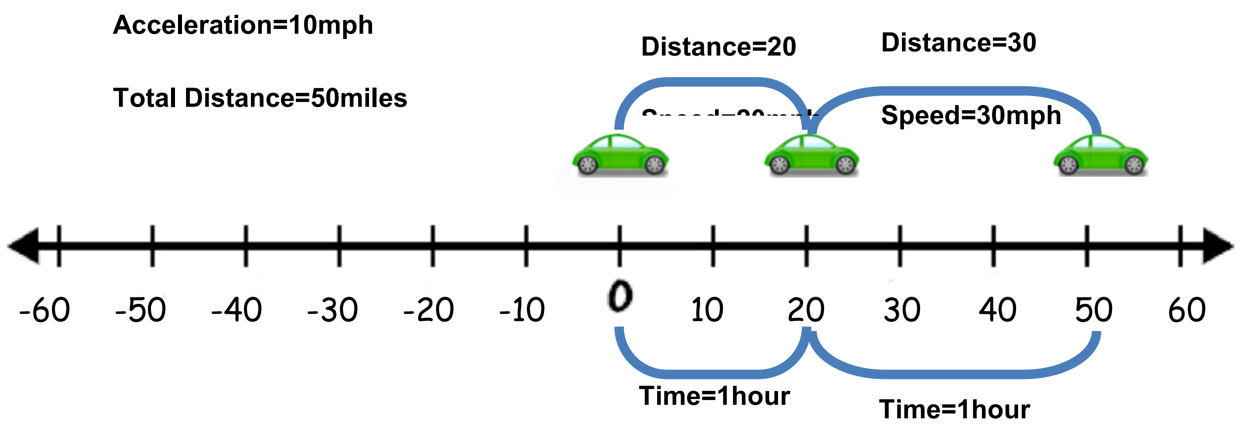
To learn about non-zero acceleration, follow the link to the page on acceleration.

***Acceleration**:*

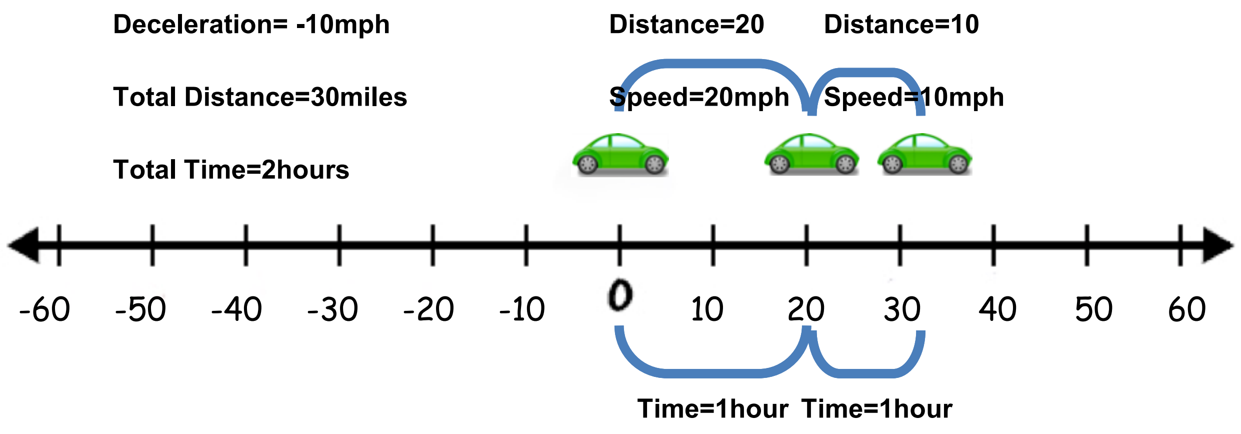
Acceleration is the *rate of change* of speed. Acceleration can either be increasing or decreasing depending on whether or not the object is increasing in speed or decreasing in speed.

Using the example of the car again, imagine that the driver of the car increased their speed linearly by 10miles/hour every hour. This means that the car has a positive constant acceleration of 10miles/hour per hour. The positive acceleration of the car means that during hour one the car increased in speed by 10 miles/hour, from 20miles/hour to 30miles/hour. Similarly, during hour two the car increased in speed from 30miles/hour to 40 miles/hour, still by 10 miles/hour.

Constant acceleration also affects the distance of an object. In the above example, the distance traveled increases by 10 miles each hour. After two hours, the car has traveled 20miles+30miles for a total of 50miles. By contrast, if the car had no acceleration and a constant speed of 20miles/hour, the car would have only traveled 40miles. The figure below illustrates acceleration as it is related to position and speed. Notice how the spaces between the positions of the car are getting bigger. This indicates the car has a positive acceleration.



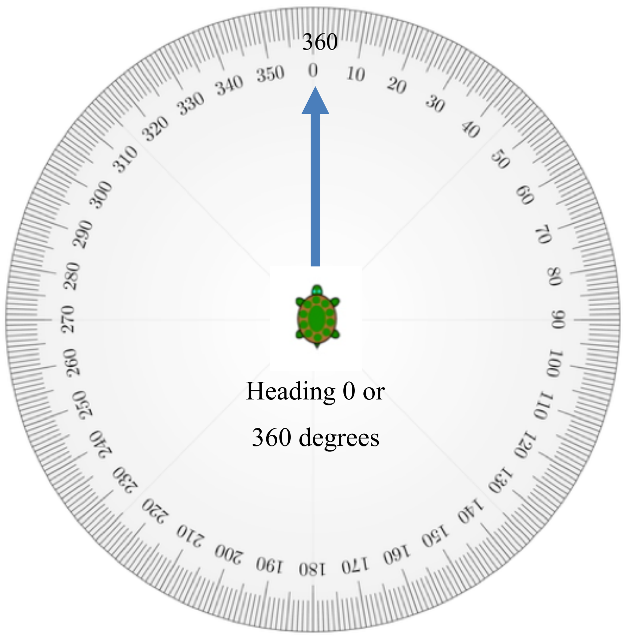
Acceleration can also be negative to indicate that an object is getting slower. Negative acceleration is called *deceleration*. Imagine the driver of the car needs to stop at a red light. As the driver applies the brakes, the speed of the car will continue to slow the car down until it reaches a stop. Imagine that the brakes of the car apply a deceleration of 10miles/minute every minute. If the car is traveling at 20 miles per hour, after two hours the car would reach a full stop. The figure below shows deceleration. Notice how the spaces between the positions of the car get smaller as the car slows down.

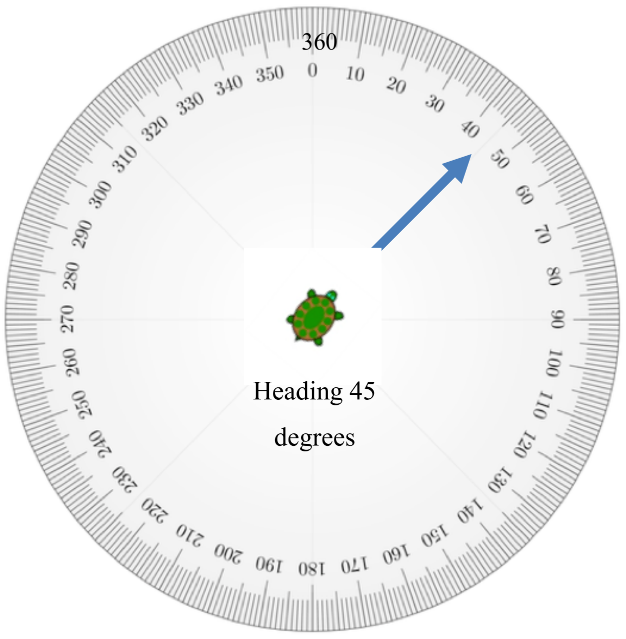


Heading of an Object

When we learned about an object's position, we thought about the position of an object as traveling in only two directions: right (positive) or left (negative). However, an object in the real world can travel in more than two directions. For our purposes, we are only going to think about two-dimensional movement although objects in the real world can move in three-dimensions.

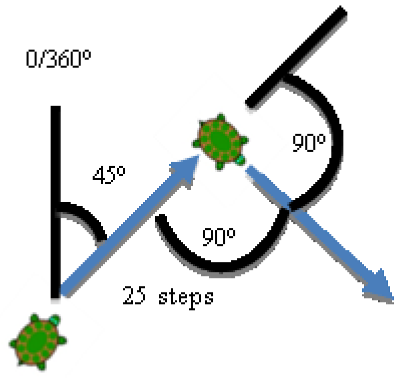
A heading is the relative position of one point with respect to another point. In the CTSiM simulations you have been interacting with, the heading of the tortoise is the direction the tortoise is moving with respect to its current position. The figure below will help explain this concept. In the figure below, the tortoise's initial heading with respect to its current position is 0/360 degrees. Note that 0/360 degrees in CTSiM refers to 90 degrees on the Cartesian coordinates. If we wanted to change the heading of the tortoise we could do that by asking the tortoise to move at a different angle. In the 2nd figure, we've asked the tortoise to move in a heading of 45 degrees. If we wanted to, we could ask the tortoise to move in any direction on the circle.





When you build your shapes in the CTSiM simulation, you will need to ask your tortoise to move in successive directions. For instance, you may want to ask your tortoise to move forward at a heading of 45 degrees and then change directions and head in a different direction. To do this, we need to think about the change in the tortoise's heading. In other words, we need to think about how many degrees we want the tortoise to turn. tortoiseThe figure below will help you think through how to ask the tortoise to change headings.

In the figure below, we first ask the tortoise to change its heading from 0/360 degrees to a heading of 45 degrees. After the tortoise changes its heading, we ask the tortoise to move forward a distance of 25 steps. After the tortoise moves 25 steps, we want the tortoise to change its heading again so that the interior angle formed by the path of the tortoise is 90 degrees. To do this we ask the tortoise to turn at an angle of 90 degrees. It is important to notice that the tortoise turns at 90 degrees from its *current* heading, not from its starting heading of 0/360 degrees. After the tortoise has turned 90 degrees, we then have asked the tortoise to continue forward for another 25 steps to produce the desired heading change and interior angle.



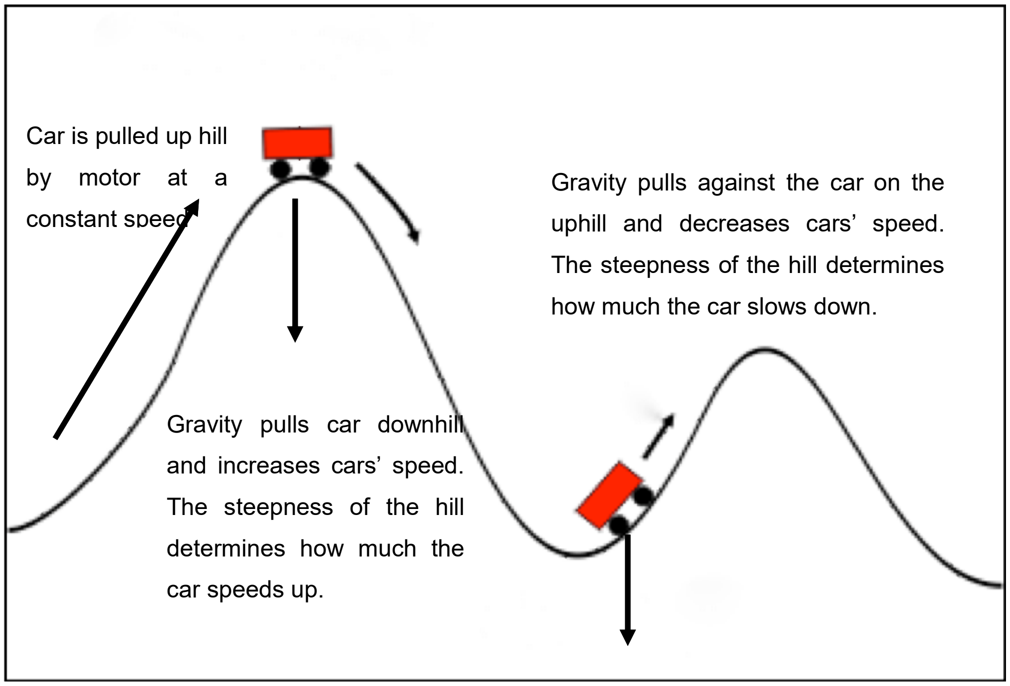
***Motion of a Roller Coaster*:**

The mechanics of motion of a roller coaster are the work of several forces. The cars on a typical roller coaster are pulled up the first hill with a chain powered by a motor. As is shown in the figure below, the cars travel up the first hill at a constant speed that is equal to the speed of the motorized chain pulling them.

After the cars make it up the first hill of the coaster track, the motorized chain stops pulling them, and gravity takes over and pulls the cars down the hill. As the cars go downhill, their speed increases gradually. This acceleration is caused by gravity. How much the speed increases as the cars travel down the hill is related to the steepness of the hill. The steeper the hill, the faster the car accelerates when going down it. This phenomenon can be seen in the first downhill of the figure below.

When the cars reach the bottom of the first hill, they start slowing down as they climb up the second hill in the figure below. Just as gravity pulls the cars downhill at an increasing speed, gravity also makes the cars decelerate as they travel uphill without the help of a motor. Gravity causes the speed of the cars to slow down when climbing uphill. Just as the steepness of a hill affects how fast the speed of the car increases as it travels down the hill, so does it affect how much a car slows down traveling up the hill. A steeper uphill slope will cause the car to slow down more than a less steep hill.

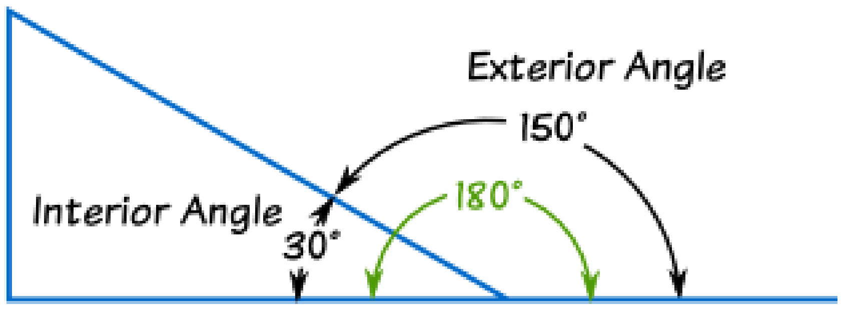
What happens when the cars are going neither downhill nor uphill, but on a flat segment? On a flat segment, there is no force that makes the cars either speed up or slow down, so the cars will travel at a constant speed. That is to say, there will be no acceleration.


Car is pulled up hill by motor at a constant speed
Gravity pulls against the car on the uphill and decreases cars' speed. The steepness of the hill determines how much the car slows down.
Gravity pulls car downhill and increases cars' speed. The steepness of the hill determines how much the car speeds up.

Drawing Polygons

A polygon is a shape with straight sides. It has the same number of sides and corners. Triangles are polygons with 3 sides and 3 corners. Squares and rectangles are polygons with 4 sides and 4 corners.

Each corner of a polygon has an interior angle and an exterior angle. The corner on the right triangle below has an interior angle of 30 degrees, and an exterior angle of 150 degrees. The interior angle and exterior angle of a corner always adds up to 180 degrees.

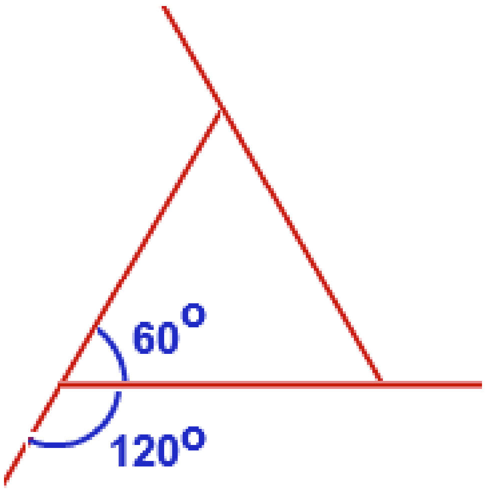


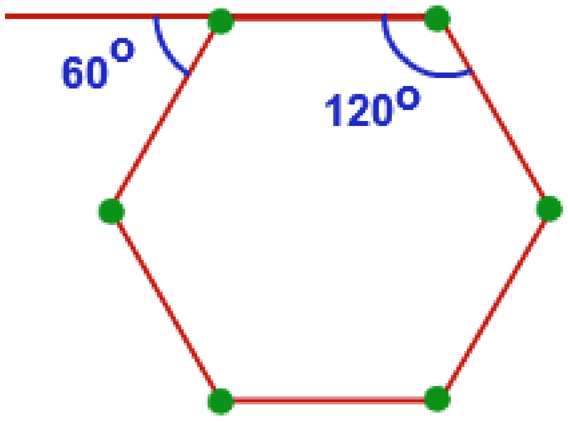
The sum of interior angles for any polygon is equal to 180 degrees * (number of sides – 2). A triangle has 3 sides, so the sum of its interior angles is 180 degrees (= 180 degrees * 1). A square or rectangle has 4 sides, so the sum of its interior angles is 360 degrees (= 180 degrees * 2).

For any polygon, if all of its sides are of equal lengths, then all of its interior angles are also equal. So each of its interior angle is equal to $\frac{180\ degrees*(number\ of\ sides-2)}{number\ of\ sides}$. For example, in an equilateral triangle, the sum of its interior angles is 180 degrees, so each interior angle is 60 degrees. In a square, the sum of its interior angles is 360 degrees, so each interior angle is 90 degrees.

In CTSiM, when we want to draw a polygon with equal sides, we need to specify how many degrees we want the tortoise to turn. The turn angle is the exterior angle, which is equal to (180 degrees – interior angle). For example, if we want to draw an equilateral triangle in CTSiM, we need to program our tortoise to turn by 180-60 = 120 degrees.

We can also calculate this exterior turn angle in CTSiM using a simple formula = 360/number of sides. For example, the turn angle for an equilateral triangle can simply be calculated as 360/3 = 120 degrees. Similarly, in a hexagon, each exterior turn angle is equal to 360/6 = 60 degrees, as you can see in the figure below.





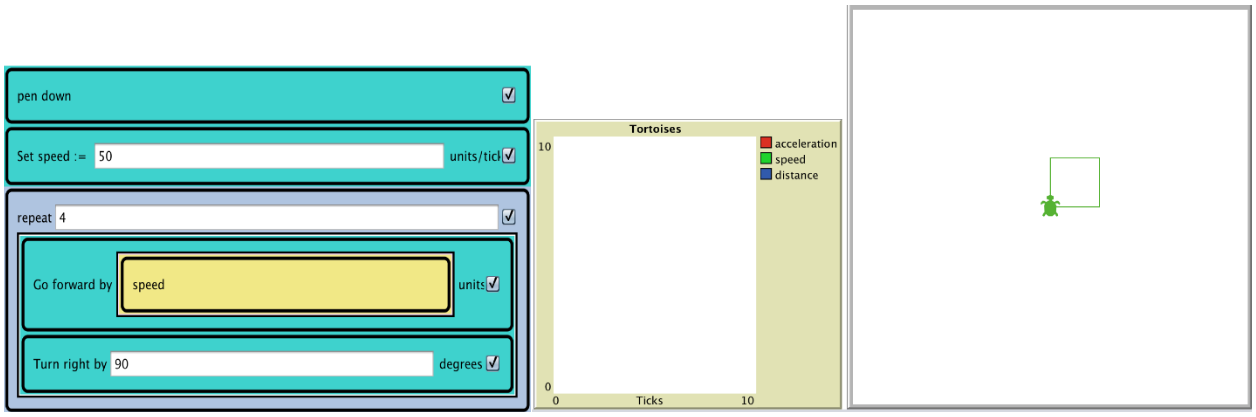
Modeling time in CTSiM simulations using ticks

We have many ways to measure time: years, months, days, hours, minutes, and seconds. We use these units to represent speed, such as 10m/s, and acceleration, such as 2m/s^2.

In CTSiM, we use ticks as our unit of time. For example, we can set the speed of a car to 10 units/tick. This means that every tick, the car moves forward by 10 units of distance. Similarly, we can set the acceleration of the car to 2 units/tick^2, meaning that every tick, the speed of the car increases by 2 units/tick.

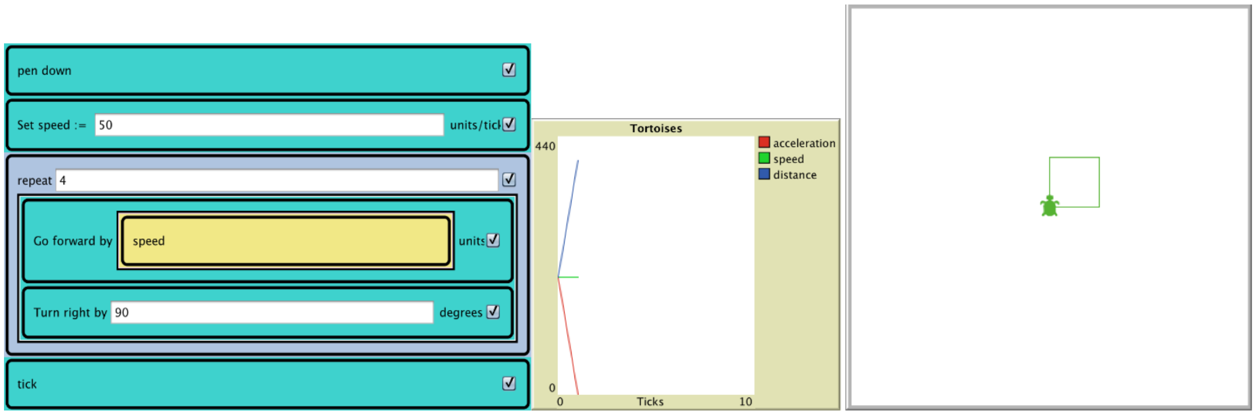
When modeling shapes in CTSiM, we also need to use the “tick” block to specify the passage of time, so that we can see the speed and distance graph when we run the model. The placement of the “tick” block in the program can affect the speed and distance graph that is generated.

Here is an example. Suppose we want to draw a square. We set the speed to 50 units/tick. We use the “Repeat” block to tell the tortoise to repeat the actions (go forward and turn right 90 degrees) 4 times. If we run this program now, we will see that the tortoise draws a square. However, there is nothing in the speed and distance graph. This is because we did not put the “tick” block in our program. So in our program, time never elapses.



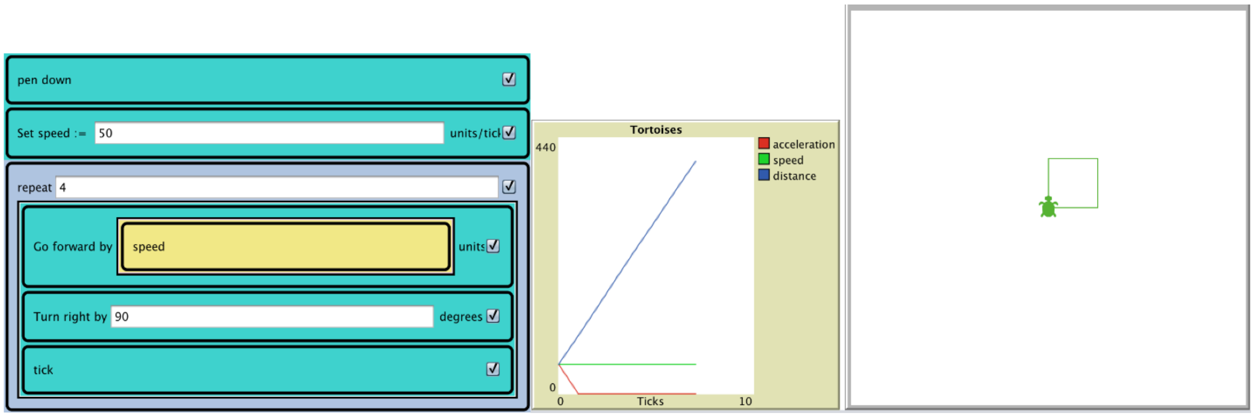
```
pen down                                   ✓
Set speed := 50                 units/tick ✓
repeat 4                                   ✓
    Go forward by speed         units      ✓
    Turn right by 90            degrees    ✓
```

If we add a “tick” block at the end of the program, outside the “Repeat” block, and run the program again, we will see that now the graph shows that the speed of the tortoise (green line) is 200 units/tick. This is because we tell the model that 1 tick elapses at the end of the program. Since the tortoise moved a total of 200 units (50 units for each side of the square), the speed is 200 units/tick.



```
pen down                                   ✓
Set speed := 50                 units/tick ✓
repeat 4                                   ✓
    Go forward by speed         units      ✓
    Turn right by 90            degrees    ✓
tick                                       ✓
```

What if we put the “tick” block inside the “Repeat” block? Now, we are telling our model that 1 tick elapses every time the tortoise goes forward and turns right once. Because the tortoise repeats the actions “Move forward by 50 and turn right by 90 degrees” 4 times, a total of 4 ticks elapse when the program finishes running. The tortoise still moves a total of 200 units in the program. So the speed of the tortoise now is 200/4 = 50 units/tick. And this is exactly what the speed graph shows us.


```
pen down
Set speed := 50 units/tick
repeat 4
    Go forward by speed units
    Turn right by 90 degrees
    tick
```

We can also think about this in another way. Each time the "Repeat" block runs, the tortoise goes forward by 50 units. We tell our model that 1 tick elapses every time a "Repeat" block is completed. So the tortoise goes forward by 50 units every tick. This means that the tortoise's speed is 50 units/tick.

A.1.2 Ecology unit

The hypertext resources used as part of the CTSiM v2 'Science Book' for the Ecology unit are included on the following pages.

Aquaponics System

A fish tank is an example of an aquaponics ecosystem. In an aquaponics ecosystem, a sustainable food production cycle is created through the interaction of the animals and plants within the system. In the fish tank, the interactions between the fish, aquatic plants and bacteria keep the water clean and the animals and plants healthy.

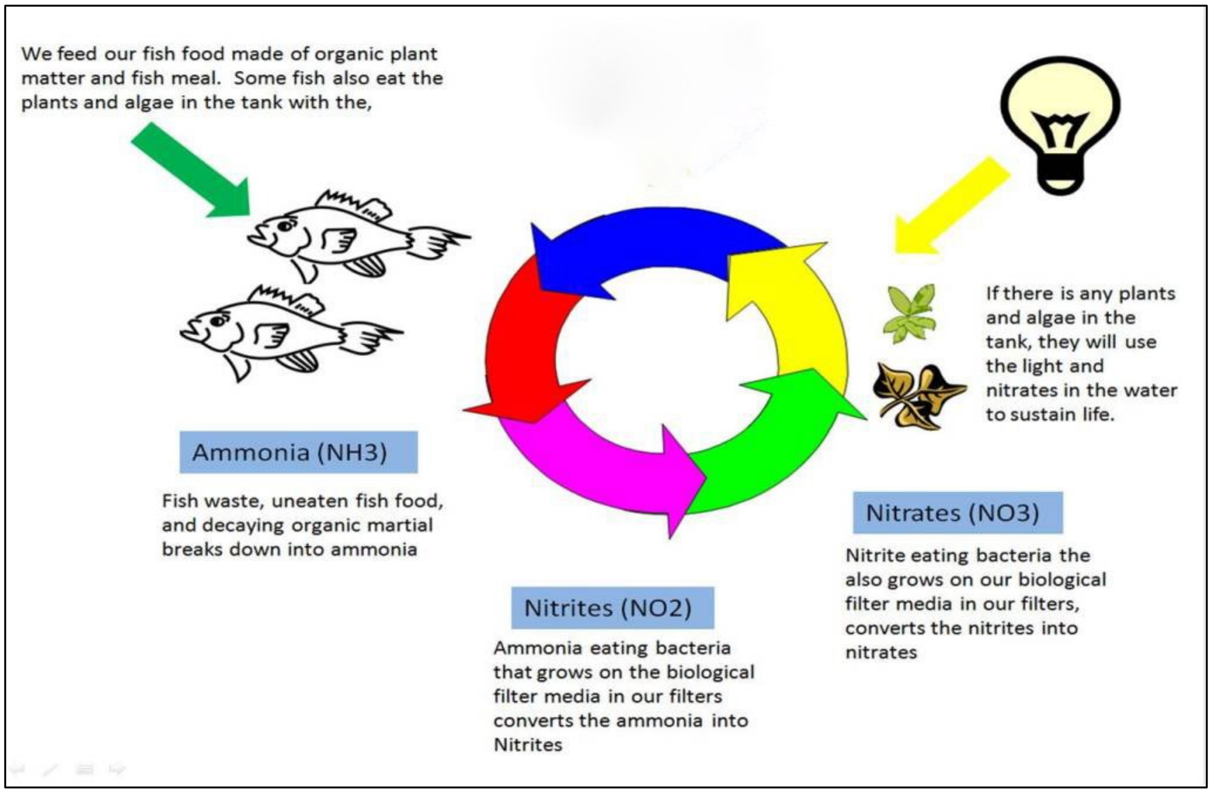
Life inside the fish tank does not need to rely on anything outside of the tank in order to sustain itself. Systems that are self-sustaining are called closed ecological systems. In a closed system, all life depends on the mutual survival of the organisms in the system. In the fish tank, the fish, aquatic plants and bacteria depend on each other to survive. If one plant or animal is removed from the tank, the other plants and animals will no longer be able to survive and will eventually die. In order to survive, any waste products and excess carbon dioxide must be converted into oxygen, food and water by other organisms in the system.

Sustainable Food Production Cycle

In ecological systems, ecosystems are considered sustainable if they are able to indefinitely maintain populations of plants and animals by consistently providing those plants and animals the resources they need to survive and reproduce. In the fish tank, the fish tank is considered sustainable when the fish, aquatic plants and bacteria are able to survive for a very long time.

The means by which the fish tank becomes sustainable is through a process called the Nitrogen Cycle. A cycle is a sequence of events that repeats itself in the same order. In an ecological cycle, all of the animals or plants that play a role in the cycle are interdependent with each other. In the fish tank, this means that the fish, aquatic plants and bacteria depend on each other for their mutual survival. If one plant or animal is removed, the cycle stops and the other plants and animals in the system die.

In the fish tank nitrogen cycle shown in the figure below, the duckweed provides food for the fish. Any food the fish is unable to metabolize is excreted as waste. Bacteria within the fish tank act upon the fish waste to produce nutrients for the duckweed. The figure below provides an example of an aquaponics nitrogen cycle.


We feed our fish food made of organic plant matter and fish meal. Some fish also eat the plants and algae in the tank with the,
If there is any plants and algae in the tank, they will use the light and nitrates in the water to sustain life.
Ammonia (NH3)
Fish waste, uneaten fish food, and decaying organic martial breaks down into ammonia
Nitrates (NO3)
Nitrite eating bacteria the also grows on our biological filter media in our filters, converts the nitrites into nitrates
Nitrites (NO2)
Ammonia eating bacteria that grows on the biological filter media in our filters converts the ammonia into Nitrites

Fish

A fish is an aquatic animal that survives in water by breathing in dissolved oxygen using its gills and breathing out carbon dioxide that gets dissolved in the water. Like all animals, fish require energy to support their biological systems and enable them to swim and stay alive. If fish have no energy, they will die. Fish gain energy by feeding on aquatic plants like the duckweed. When fish are hungry, they swim towards the nearest aquatic plants and eat. When they eat, their energy increases, and the food source decreases. If fish are hungry but cannot find aquatic plants, they cannot eat. When fish are not hungry, they just swim randomly from one area of the fish tank to the other. Swimming decreases fish energy. It does not matter whether fish swim randomly or in a fixed direction towards food, energy is always used up in swimming.

Like all other organisms, fish produce waste. Any food that cannot be converted into energy for the fish is eliminated from the body in the form of metabolic waste. Fish waste contains high amounts of a chemical called ammonia. If fish waste is allowed to accumulate in the fish tank, the high amount of ammonia will begin to pollute the water and over time can make the water toxic for the fish. Water that is toxic is unhealthy for the fish and can make them sick. If the amount of toxic chemicals from the fish waste reaches a poisonous level, the fish will die even if the fish have access to food and oxygen. Conversely, if the level of toxic chemicals from the fish waste is low, the fish will remain healthy.

Below you will find pictures of different types of fish you may find in a fish tank.

Common Goldfish

Neon Tetra

Gourami

Beta

Aquatic Plants

An aquatic plant is a plant that lives in or near water. Duckweed is a type of aquatic plant that many fish use as a food source. Like all plants, aquatic plants also need to breathe. They utilize photosynthesis to convert carbon dioxide dissolved in the water into dissolved oxygen.

Plants also need energy. Plants on land can obtain necessary energy from soil and water in the form of fertilizers. Many fertilizers contain a chemical called nitrate that the plant can use as a food source to gain energy. Aquatic plants like duckweed also obtain energy from nitrates and the photosynthesis process. Unlike fish, which needs to be hungry and have food available in order to eat, aquatic plants consume food whenever food is available, and they do not produce waste. Consuming nitrates increases plant energy and decreases the food source, that is the amount of nitrates. Plants use this energy for reproduction and other essential life processes. Plant can reproduce only if they have enough energy, since reproduction causes their energy levels to decrease a lot. If plants have no energy left, they die.

Below you will find examples of several aquatic plants you may see in a fish tank.

Duckweed

Egeria

Amazon Swords

Hydrilla

Breathing

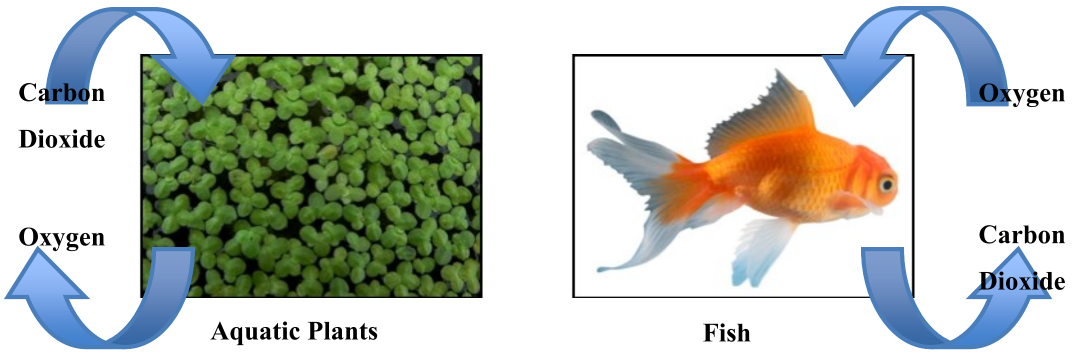
In living organisms, breathing is process of gas exchange within the body of the organism. As one gas in taken into the body, another is removed. Which gases enter and exit the organism depends on the type of organism.

In animals, breathing is a process that moves oxygen over the respiratory organs (such as lungs or gills) of an animal. Breathing does two things for an animal: 1) it gives animals the oxygen they need to stay alive and support necessary bodily functions, and 2) helps the animal remove harmful carbon dioxide from their bodies.

Plants also 'breathe', although the mechanism of breathing is different for plants than it is for animals. In plants, carbon dioxide is taken into the plant through small holes called stoma. This carbon dioxide is then converted into oxygen and released in a process called photosynthesis.

Oxygen and carbon dioxide both play important roles in animal and plant health. In animals, oxygen is an essential component of survival. Without oxygen, animals would not be able to perform necessary functions and would die. Similarly, plants would die in the absence of carbon-dioxide.

For example, in a fish tank, the fish will die in the absence of dissolved oxygen. Fish breathe in the dissolved oxygen and breathe out carbon-dioxide which gets dissolved in the water. This decreases the amount of dissolved oxygen and increases the amount of dissolved carbon-dioxide in the fish tank. Aquatic plants like the duckweed breathe in this dissolved carbon-dioxide and breathe out oxygen which is needed by the fish. This increases the amount of dissolved oxygen and decreased the amount of dissolved carbon-dioxide in the fish tank. Together, the breathing processes of the fish and the duckweed keep the amount of dissolved oxygen and carbon-dioxide in the fish tank in balance. Thus, the respiration cycle is important in maintaining balance in a fish tank and keeping its organisms alive. The figure below summarizes which gases plants and animals in the fish tank take in, and which gases are expelled.



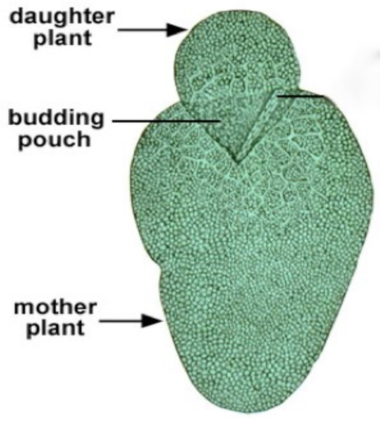
Water

Water is an essential component of life on Earth. Scientists currently do not know what percentage of life lives in water but the estimates are anywhere between 30 to 70% of life lives in water.

In order for organisms to thrive in water, the water must be conducive to life. Most aquatic organisms, such as freshwater fish in the fish tank, require access to oxygen and water that is clean to keep them healthy and alive. Oxygen is produced through aquatic plants. Clean water is water that is low in both natural and artificial toxins. Artificial toxins can enter water through manmade pollution; however, natural toxins occur as well. Waste produced by aquatic organisms like the fish contains chemicals like ammonia. If those chemicals like ammonia are allowed to accumulate, the water can become poisonous. Water that is too high in toxins is unhealthy for aquatic organisms like the fish and can lead to sickness and finally death. Thus, bacteria play an important role in the fish tank by keeping the water clean and pollution-free.

Reproduction

Reproduction is a fundamental feature of all life and is the process by which new individual organisms are produced from parent organisms. In fish, new fish are created through sexual reproduction, which is the combining of genetic material of two parent organisms. In plants, new plants are created through either sexual (two parents) or asexual (one parent) reproduction. In the fish tank, duckweed reproduces both asexually through budding (shown below) as well as sexually.



Budding in Duckweed

The process of reproduction requires a great deal of energy and only healthy organisms are able to reproduce. If an organism does not have enough energy, it will not be able to reproduce. Reproduction decreases the organism's energy significantly.

The length of time an organism needs to reproduce is highly variable and depends on the type of organism. For instance, humans require 9 months to produce an offspring. Fish, on the other hand, may hatch from their eggs in 1-3 weeks. Even faster than fish, asexual plant reproduction may require only a few days to produce a functional new plant. The fish tank simulation in CTSiM provides only a brief glimpse of life in the fish tank. For this reason, the duckweed appears to reproduce while the fish do not. This is because the simulation is only showing a few days in a fish tank rather than several weeks.

Bacteria also reproduce faster than fish and can reproduce as long as they have enough energy. When they reproduce, their energy decreases significantly like in the case of the duckweed.

Bacteria

Bacteria are very small organisms that are generally not visible to the naked eye. Some varieties of bacteria are harmful and cause diseases in humans. But, some types of bacteria are good and useful. For example, we find two types of bacteria in a fish tank – Nitrosomonas bacteria and Nitrobacter bacteria. They are both needed to maintain a healthy fish tank and fish will die in their absence. Let us see why these two types of bacteria are so important in the fish tank.

Nitrosomonas is a type of bacteria that gains energy by consuming the ammonia contained in fish waste. They convert the ammonia consumed to another chemical called nitrites. Thus, Nitrosomonas is important to the fish tank because they clean the tank by decreasing toxic ammonia that can be poisonous to the fish. .

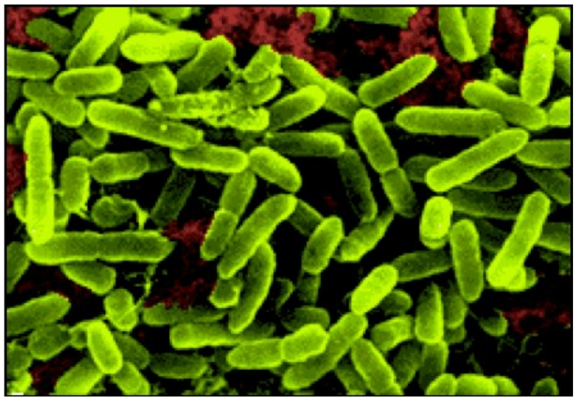
But, Nitrosomonas cause amount of nitrites in the fish tank to increase, and nitrites are also toxic and can be poisonous for the fish. The Nitrobacter bacteria help by removing the nitrites from the fish tank. Nitrobacter gain energy by consuming the nitrites produced by the Nitrosomonas bacteria. They convert the nitrites into a new chemical called nitrates. In a fish tank, these nitrates serve as a food source for the aquatic plants like duckweeds. Without these bacteria, fish waste would keep increasing in the fish tank making the water poisonous for the fish, and duckweed would have no food available.

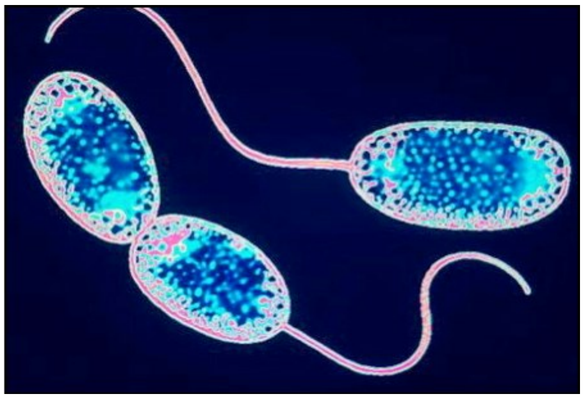
Hence, we can see how all the species in a fish tank depend on each other. Duckweed acts as food for fish. Fish waste contains ammonia which is consumed by the Nitrosomas bacteria and converted into nitrites. Nitrites are consumed by the Nitrobacter bacteria and converted into nitrates. These nitrates act as food for the duckweed.

Bacteria can also feel hungry like other organisms. For example, Nitrosomonas eat when they are hungry and have ammonia available. Similarly, Nitrobacter eat when they are hungry and have nitrites available.

Both Nitrosomonas and Nitrobacter swim around randomly; they don't swim in any particular direction. When they swim, their energy decreases.

Nitrosomonas and Nitrobacter can both reproduce as long as they have enough energy. When they reproduce, their energy decreases significantly. Nitrosomonas and Nitrobacter die when they have no energy left.

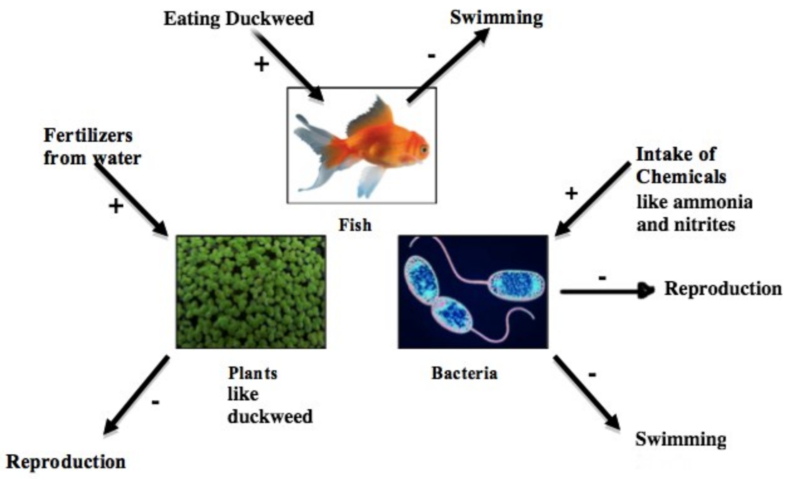

Nitrosomonas Bacteria


Nitrobacter Bacteria

Energy

All living organisms rely on external (outside of the body) sources of energy to stay alive. In animals like mammals, birds and fish, energy comes from the meat or plant food that the animal eats. For example, fish gain energy by eating aquatic plants like duckweed. In organisms like bacteria, chemicals such as ammonia and nitrites can be used as form of energy. For example, the nitrosomonas bacteria gain energy by consuming ammonia, while the nitrobacter bacteria gain energy by consuming nitrites. Plants obtain energy through fertilizers in the soil and water. For example, duckweed gain energy by consuming a chemical called nitrates which is present in fertilizers.

Energy provides an organism the means to perform necessary functions such as moving and reproducing. For example, fish and bacteria use up their energy when they swim around. Plants like duckweed and bacteria use a lot of energy in reproduction. Reproduction generally causes a greater decrease in energy than other life processes. Without enough energy, an organism would not be able to survive and would die. The figure below summarizes which process give energy to the organisms in the CTSiM fish tank and which cost them energy.



Glossary of Terms

- Aquaponics: A sustainable food production system that combines aquaculture (raising aquatic animals such as fish, snails or crayfish) with hydroponics (growing plants in water) in a symbiotic environment.

- Dissolved Oxygen: the amount of oxygen that is carried within a medium. In the case of the fish tank, the level of dissolved oxygen is that amount of oxygen that has been incorporated into the water.

- Interdependence: A relationship wherein each member – either plant or animal – is mutually dependent on other members.

- Metabolize: A chemical transformation that happens within the cells of living organisms. Metabolism is the process whereby food is transformed into usable energy for the organism.

- Nitrogen Cycle: The nitrogen cycle is the process whereby nitrogen is converted into its various chemical forms. In the fish tank, ammonia, nitrite and nitrate are all chemical forms of nitrogen.

- Organism: An organism is an individual form of life, such as plants, animals, protists, bacterium, or fungi. Organisms have bodies made up of organs, organelles, or other parts that work together to carry on the various processes of life.

- Prokaryote: A group of organisms whose cells lack a membrane bound nucleus. Organisms whose cells have a nucleus are called eukaryotes.

- Photosynthesis: a process used by plants to convert light energy – normally from the sun – into chemical energy that can be used to fuel the plants' activities.

- Resource: a substance or object required by a living organism for normal growth, maintenance and reproduction.

- Sustainable: the capacity to endure. In ecology, sustainability refers to the capacity of biological systems to remain diverse and productive over a long period of time.

A.2 Programming Guide

The hypertext resources used as part of the CTSiM v2 ‘Programming Guide’ are included on the following pages.

Modeling a science topic in CTSiM

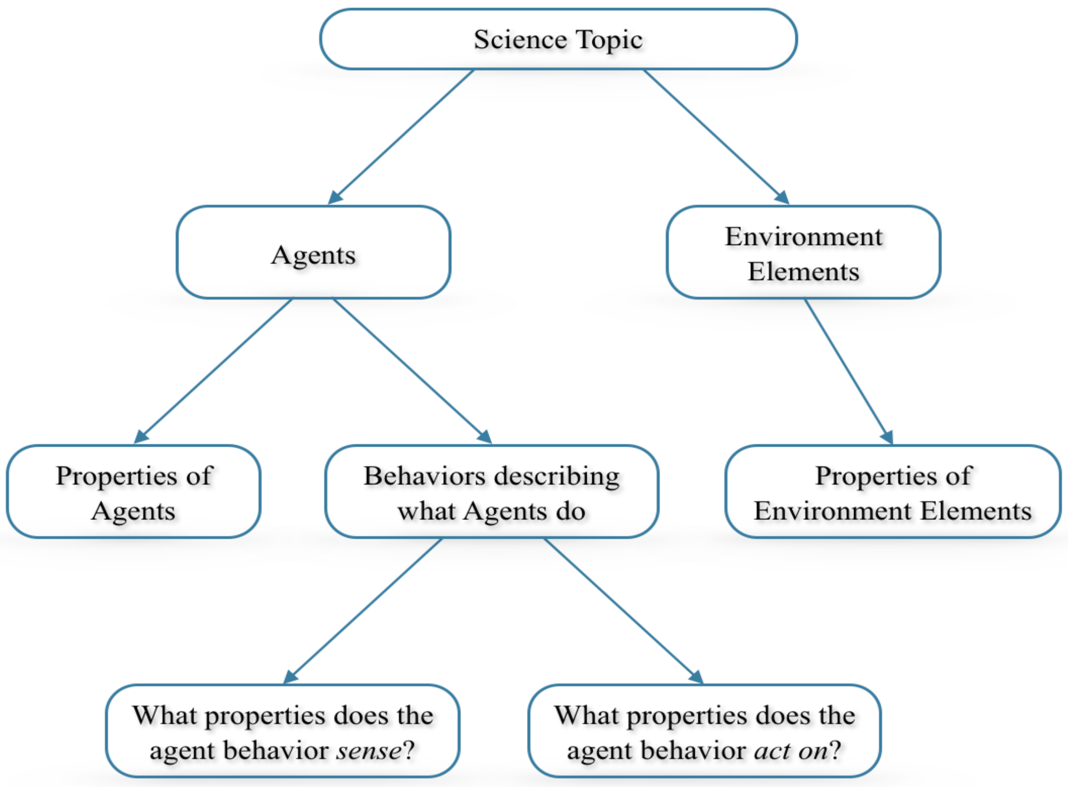
A good way to think of how to build a model for a science topic starts with identifying the entities that are part of the science topic and actively do something in the topic. For example, if we are building a model for the traffic outside your school, cars, school-buses and pedestrians would be examples of such entities.

These entities are called agents. Once we have identified the agents in the model, we need to describe properties of the agents that are of interest to us, and what the agents do. We refer to each feature describing an agent as an agent property, and each thing that an agent does as part of an agent’s behavior. In other words, an agent may have more than one behavior. For example, if we consider the school-bus agent, we could describe it using properties like speed, number of seats, and color. The bus would have different behaviors like ‘Move at a constant speed’, ‘Slow down’, and ‘Speed up’.

After specifying the properties and behaviors for each agent, we need to describe what happens in each agent behavior. An agent behavior describes how the agent interacts with other agents and also with its surrounding environment. The entities in the environment are called environment elements and features describing them are known as environment element properties. An agent’s behavior can act on and change its own properties or properties of other agents or environment element properties. The behavior might also depend on different agent and environment element properties and will have to sense those properties.

For example, in the bus agent’s ‘Slow down’ behavior, the speed of the bus decreases and we say that the ‘speed’ property of the bus is acted upon. But, when does the speed of the bus decrease? It could decrease when the bus is approaching a red light or stop sign, or when the road is icy, or when there is a car moving slowly in front of it. Thus, the ‘Slow down’ behavior for the bus agent will have to sense environment properties like traffic-light color and road condition and agent properties like car speed.

The chart below shows what we have discussed above.


Science Topic
Agents
Environment Elements
Properties of Agents
Behaviors describing what Agents do
Properties of Environment Elements
What properties does the agent behavior *sense*?
What properties does the agent behavior *act on*?

Read the pages on agents, environment elements, properties, and behaviors for specific examples and more information on how to describe a science topic in CTSiM.

Agents

When modeling a science topic, we need to first identify the entities that actively do something in the topic. These entities are called agents.

For example, suppose we want to describe the traffic on the road in front of your school. Here, the cars on the road are agents because they actively move at different speeds and in different directions. If there are school-buses, bicycles and pedestrians on the road, they are other types of agents that participate in our traffic model. So we see that a model can have one agent or multiple agents.

Agent type 1: car

Agent type 2: school-bus

Agent type 3: bicycle

Agent type 4: pedestrian

Once we have identified the agents in the model, we need to describe features of the agents and what the agents do. We refer to each feature describing the agent as an agent property, and each thing that the agent does as an agent behavior. Agents interact with other agents, as well as other things around them that are not agents. These surrounding entities are called environment elements.

Environment elements

Unlike agents, there are entities in a topic that do not actively do anything. We call them environment elements.

For example, suppose we want to describe the traffic on the road in front of your school. Here, the road is an example of an environment element. A model can have more than one type of environment element. In our traffic model, traffic lights can be considered as another type of environment element.

Environment element type 1: road

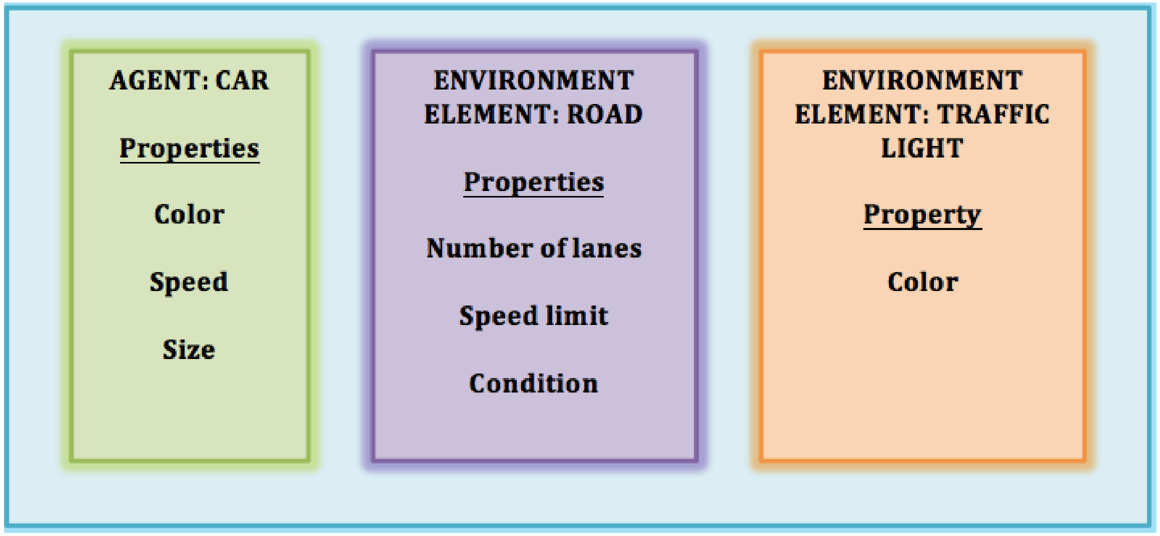
Environment element type 2: traffic light

Once we have identified the types of environment elements in a topic, we need to specify features to describe each of the types. We refer to each feature describing an environment element as an environment element property.

Properties – Agent properties and Environment element properties

When describing a science topic, we need to specify properties for all the entities in the topic. These properties help describe an entity and distinguish it from other entities.

In any topic, there are broadly two kinds of entities - agents which actively do something, and environment elements which surround the agents and do not actively do anything. Both agents and environment elements have properties. For example, if we are describing the traffic on the road, cars are a type of agent, and the roads and traffic lights are types of environment elements. Cars' properties include color, size, and speed. Road's properties would be number of lanes, speed limits and condition (ice or dry, for example), and an example of a traffic light's property is color (for example, red, yellow, and green).



AGENT: CAR

Properties

Color

Speed

Size

ENVIRONMENT ELEMENT: ROAD

Properties

Number of lanes

Speed limit

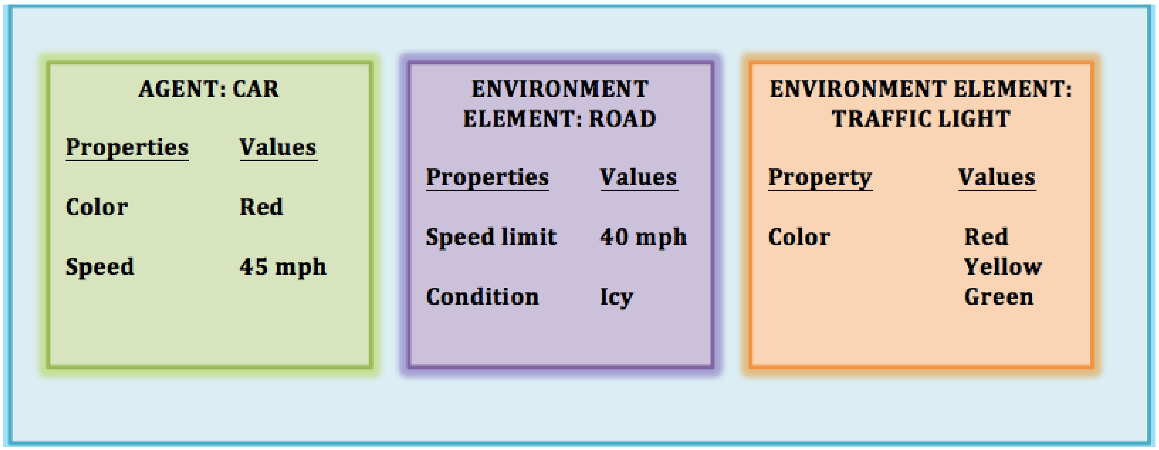
Condition

ENVIRONMENT ELEMENT: TRAFFIC LIGHT

Property

Color

Most properties have values. For example, if a red car is running at 45 mph, we say that its "color" property has the value "red", and its "speed" property has the value "40". Similarly, if a road is icy, we say its "condition" property has the value "icy".



AGENT: CAR

Properties	Values
Color	Red
Speed	45 mph

ENVIRONMENT ELEMENT: ROAD

Properties	Values
Speed limit	40 mph
Condition	Icy

ENVIRONMENT ELEMENT: TRAFFIC LIGHT

Property	Values
Color	Red Yellow Green

Once we have set the value of a property, we can refer to the property using just its property name, rather than its value. For example, we can set the speed of our car by saying "set speed = 50". Now we can just say "forward (speed)" which will make the car move forward 50 units. When we run our program, the computer will automatically look at the value of the property "speed" and move forward at that speed.

Agent behaviors

Agents are entities that actively do something. We call each thing that an agent does a behavior. When we are specifying an agent's behaviors, we separate different things that the agent does into different behaviors. Continuing our previous example with the traffic scenario, let us think of examples of a car's behaviors in a traffic model. A car stops when it sees a stop sign or when the traffic light is red. When the traffic light is green, the car moves forward. A car will slow down if the road is icy or if there is a slow moving vehicle in front of the car. In each of these scenarios, do you recognize the car's behaviors? The car's behaviors include stopping, moving forward, and slowing down.

AGENT BEHAVIOR:
CAR BEHAVIOR

Stop

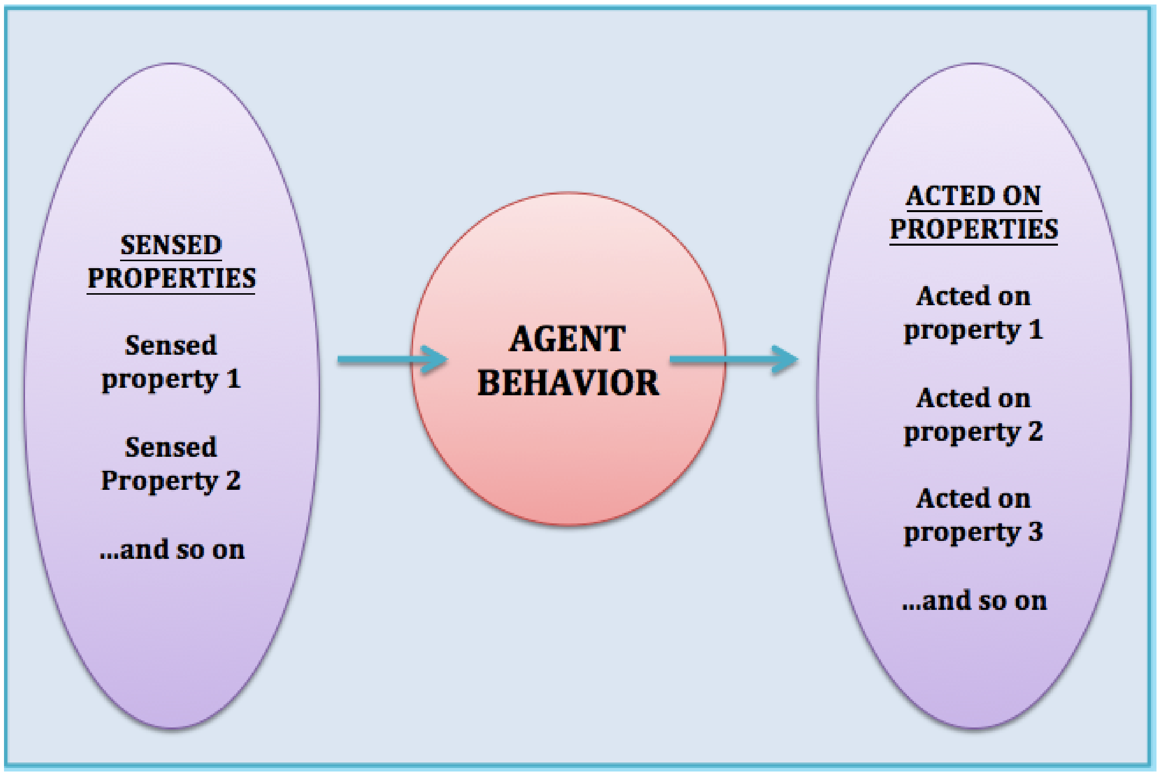
Move forward

Slow down

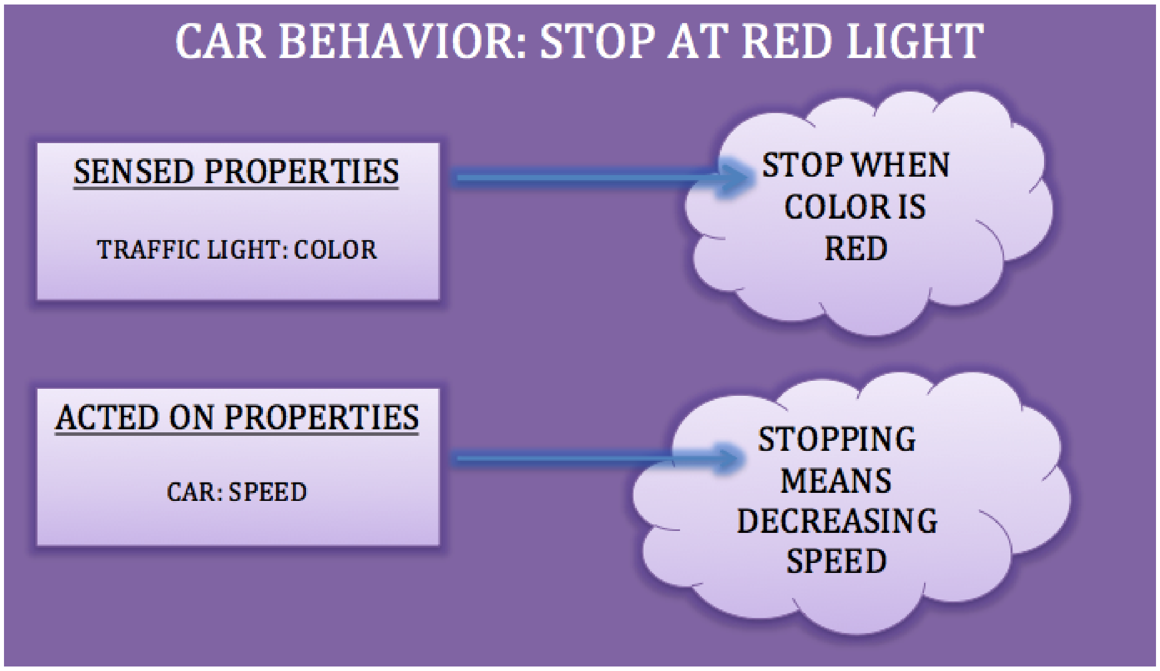
Once you specify an agent's behaviors, you need to think about each behavior in terms of the properties it needs to sense and the properties it needs to act on. Read the next page for examples on how to do this.

Modeling agent behaviors using sensed and acted on properties

Once you specify an agent's behaviors, you need to think about what are the things a behavior needs to know and what are the things it changes. A behavior changes or acts on certain properties and senses certain other properties in order to make these changes. An agent behavior can both sense and act on own properties, as well as other agent properties or environment properties.



For example, think about a car's "Stop at red light" behavior. When the car senses that the traffic light color is red, it slows down and finally stops by changing or acting on its speed. The figure below shows how to represent the car's behavior in terms of its sensed and acted on properties.

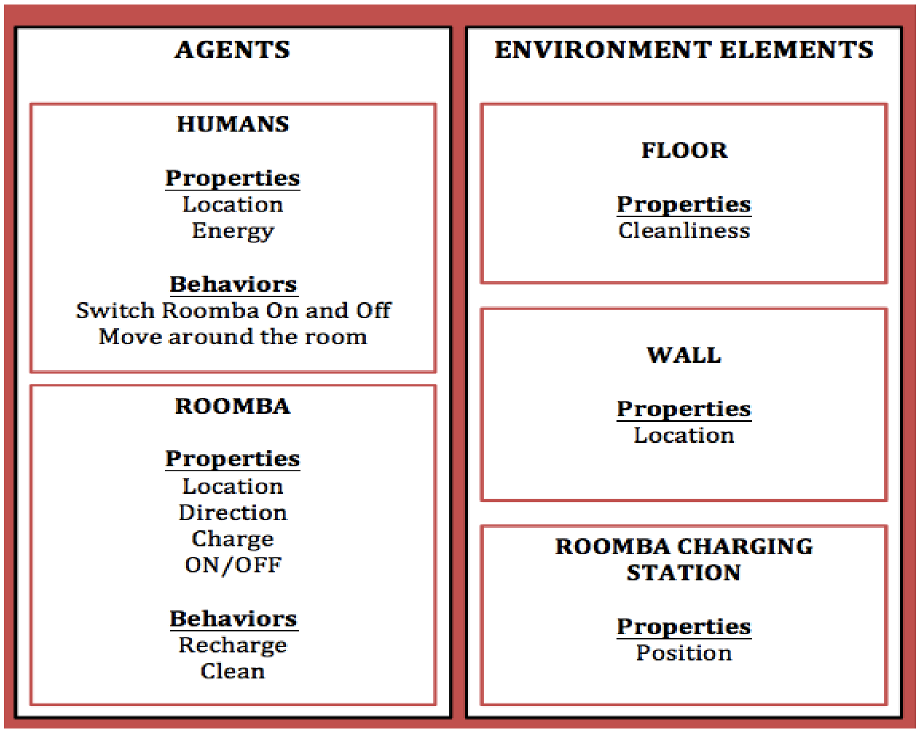


Examples of Conceptual Modeling for Different Topics

Let's think of another example from real life and try to model it conceptually using the agent framework we read about in the previous pages. Consider a scenario that involves a human and the vacuum-cleaning robot, Roomba, in a room. Roomba moves around the room to clean floors and carpets. Roomba is also able to change direction when it encounters humans and walls in the room. When Roomba's charge runs out, it goes to a charging station positioned on a wall to recharge itself. Although Roomba can clean, move and recharge by itself, humans are responsible for turning Roomba on and off.

To model this scenario, we can consider humans and Roomba as two types of agents with different sets of behaviors, and we can consider the floor, the walls and Roomba's charging station as environment elements. First, we need to think about the properties of each agent and environment element in this scenario. Examples of human agents' properties are location and energy, while examples of the Roomba agent's properties are location, direction, charge, and its On/Off switch. Examples of the floor's property could be its cleanliness, the wall's property its location, and Roomba's charging station's property its position.

Next, we need to think about the agent behaviors. Human behaviors are switching Roomba on and off, and moving around the house. Roomba's behaviors are cleaning and recharging. We could represent this conceptual model as shown below.



Now that we have specified the agent behaviors, let us try to describe all the behaviors in terms of what the behavior needs to sense and what properties it acts upon. When a human notices that the floor is dirty, what happens? Well, the human senses the cleanliness property of the floor and acts on Roomba's On/Off property by turning it on. Then, when Roomba senses its own On/Off switch property has turned On, it cleans the room, which means it acts upon its own location and direction properties as well as the floor's cleanliness property. What else does Roomba sense while it is cleaning? Humans' and walls' locations! Humans move around the house, acting upon on their own location and their energy. So when Roomba senses that a human or a wall is in its way, it acts upon its direction property by changing direction. Finally, when Roomba senses that it has run out of charge, it will act upon its own charge property by recharging itself. Also, it will sense where the charging station is located and act upon its location property by moving to the charging station.

HUMAN BEHAVIOR: SWITCH ROOMBA ON AND OFF

SENSED PROPERTY

FLOOR: CLEANLINESS

ACTED ON PROPERTY

ROOMBA: ON/OFF

HUMAN BEHAVIOR: MOVE AROUND THE ROOM

SENSED PROPERTY

WALL: LOCATION

ACTED ON PROPERTIES

HUMAN: LOCATION
HUMAN: ENERGY

ROOMBA BEHAVIOR: CLEAN

SENSED PROPERTIES

ROOMBA: ON/OFF
HUMAN: LOCATION
WALL: LOCATION

ACTED ON PROPERTIES

FLOOR: CLEANLINESS
ROOMBA: LOCATION
ROOMBA: DIRECTION

ROOMBA BEHAVIOR: RECHARGE

SENSED PROPERTIES

ROOMBA: CHARGE
ROOMBA CHARGING STATION: POSITION

ACTED ON PROPERTIES

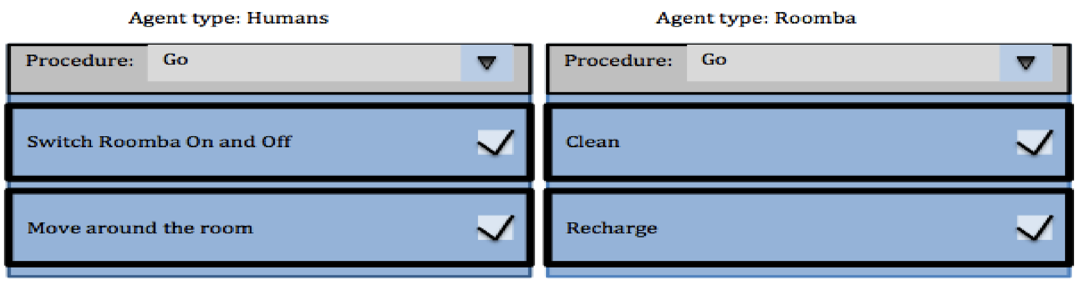
ROOMBA: CHARGE
ROOMBA: LOCATION

Programming an agent model

While writing a program to model a science topic, we need to follow two steps. First, we write a procedure or sub-program to describe each agent behavior. While writing the procedures, we will need to express how the agent interacts with other agents and the environment in that procedure. To express these, we will need to learn about some computer-science commands like "Repeat" and "When…Do…Otherwise do…" Then, we need to specify each of the procedures under a "Go" procedure. The "Go" procedure is where our program starts running and only the procedures specified there will be included while generating simulations for our program.

Let us consider the scenario on the previous page with humans and Roomba, the vacuum-cleaning robot in a room. Roomba cleans the room and recharges its own battery. Humans do not have to clean the room but they move around and they switch Roomba on and off as needed. So, if we want to write a program for this scenario, we will need to write 2 procedures for Human agents – one called "Switch Roomba On and Off" and another called "Move around the room", and 2 procedures for the Roomba agent – one called "Clean" and another called "Recharge".

We also need to write a main "Go" procedure for each agent that calls the different procedures for that agent. In our example, both Humans and Roomba will have a main "Go" procedure that calls their respective procedures, as shown in the figure below.


Agent type: Humans

Procedure: Go

Switch Roomba On and Off ✓

Move around the room ✓

Agent type: Roomba

Procedure: Go

Clean ✓

Recharge ✓

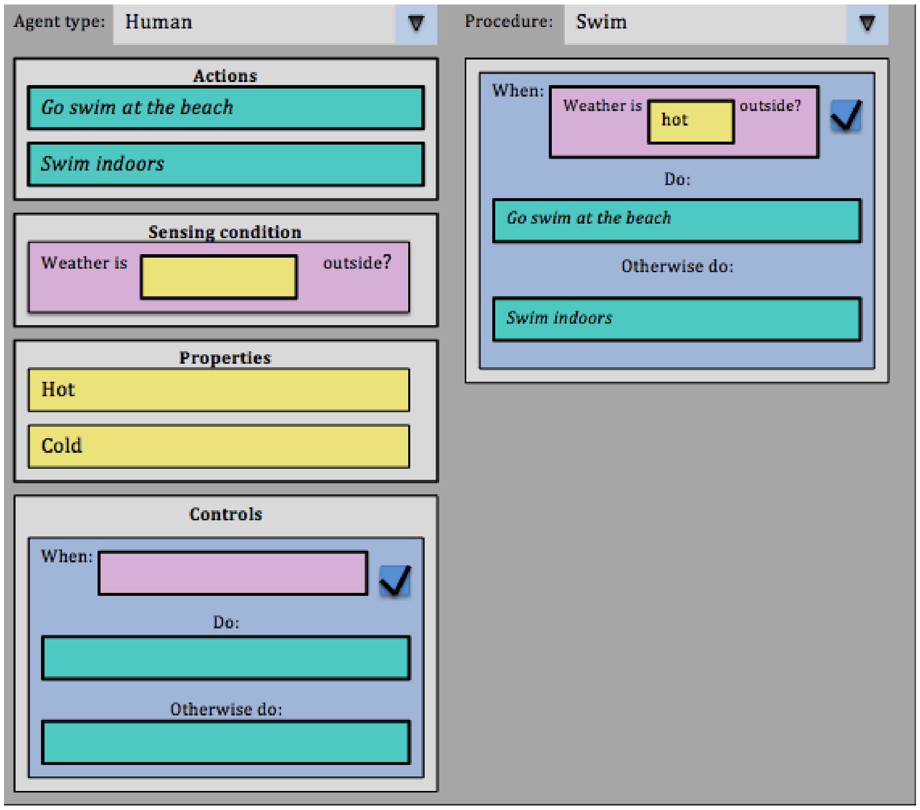
For example, for describing the "Switch Roomba On and Off" procedure for humans, the behavior will check if the floor is clean or dirty. If the floor is dirty, humans will turn on Roomba's On/Off switch. Otherwise, they will turn it off. We can represent this using the "When…Do…Otherwise…" block.

Similarly, the “Recharge” behavior for Roomba will use the “When…Do…Otherwise do…” block to sense if its battery charge is low. When the battery charge is low, Roomba can use the “Repeat” block to specify the following action multiple times: “Battery charge is low, so move backwards” till it reaches its charging station.

So, how do you write a program for each agent procedure? You need to drag and drop different blocks which will be provided to you and arrange them in a way that makes sense for the procedures. The blocks can be of different types like Agents, Agent Properties, Actions, Sensing conditions, Sensing amounts, Controls, and Chemicals. Each type of block is generally color coded differently. For example, all Sensing Condition blocks may be purple, while all Action blocks may be blue in color. So, what do each of these types of blocks express?

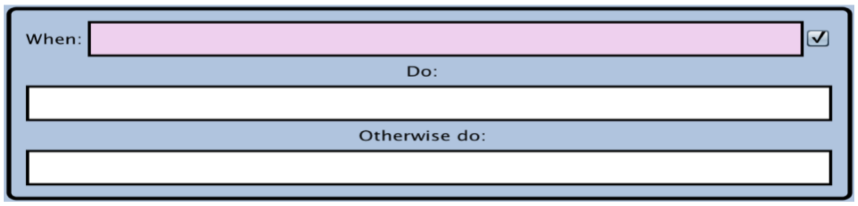
Block type	Description	Examples
Agents	Entities that actively do something	Humans, Cars
Agent properties	Features describing an agent	Human energy, Car speed
Actions	Things that agents do	Move forward; Stop, Eat a cookie
Chemicals	A form of matter that has constant chemical composition and properties	Dissolved Oxygen; Dissolved Carbon Dioxide (CO2)
Sensing conditions	Conditions for sensing an agent’s or environment’s properties	“Is it sunny outside?”; “Is thirsty?”
Sensing amounts	Property values sensed	Number, Color, Weight, Time
Controls	Used to write programs and express conditionals, iterations, etc.	“When…Do…Otherwise”; “Repeat”

When you’re dragging and dropping blocks, you need to remember that all blocks cannot be randomly dropped into other blocks. For example, you cannot put an Action block into the “When” block of a "When…Do…Otherwise" block, and you cannot put a Sensing Condition block into the "Do" part of a "When…Do…Otherwise" block. You also can only drop one block into another block if both of those blocks are of the same color. For example, in the figure below, the blocks in the left pane were provided and they have been dragged to the right pane and arranged to describe a swim procedure for human agents. Notice how you can only drop a pink sensing condition block into the pink "When" block, and you can only drop a yellow Properties block into the yellow block within the sensing condition block.



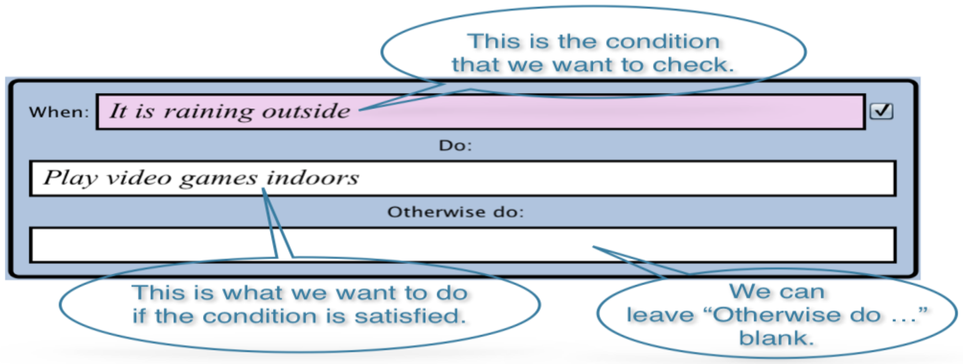
Representing sense-act processes using "When… Do… Otherwise do…"

We use the "When … Do … Otherwise do …" block to measure a property, and act in one way or another based on the measurement. Basically, we represent a "sense-act" process—we "sense" whether a property satisfies a certain condition, and decide to "act" in some way based on what we sense.



An empty "When … Do … Otherwise do …" block

Imagine you are told to play video games indoors if it is raining outside. We can express this using the "When … Do … Otherwise do …" block like this:


This is the condition that we want to check.
When:
It is raining outside
Do:
Play video games indoors
Otherwise do:
This is what we want to do if the condition is satisfied.
We can leave "Otherwise do …" blank.

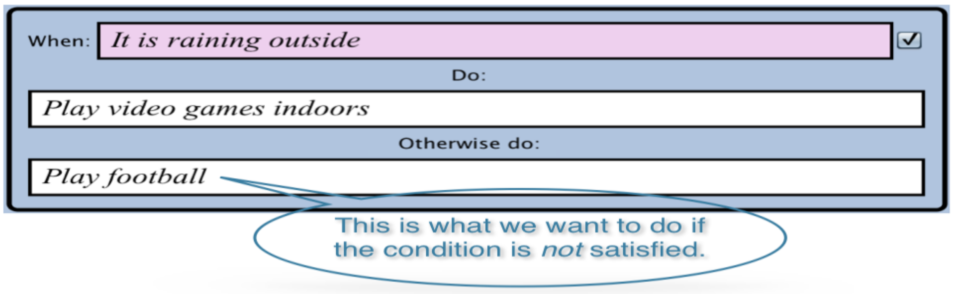
When it is raining outside, play video games indoors.

As we can see from this example, we first check whether a condition is satisfied. In this case, we sense whether it is raining. When the condition is satisfied, we act in some way. In this case, the decision is to "play video games indoors."

Notice that we can leave "Otherwise do …" blank if we don't want to do anything when the condition is not satisfied. However, we cannot leave "Do" blank.

A lot of times, we want to do something when a condition is satisfied, but do something else when the condition is *not* satisfied. For example, we may decide to stay indoors when it is raining. Otherwise (when it's not raining), we may decide to go play football.

Now we can express this using "When … Do … Otherwise do …" like this:


When: *It is raining outside* ☑

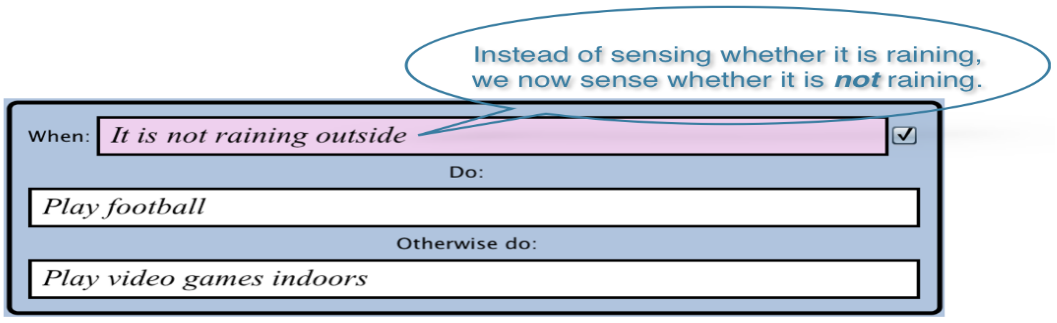
Do: *Play video games indoors*

Otherwise do: *Play football*

This is what we want to do if the condition is *not* satisfied.

When it is raining outside, play video games indoors. Otherwise, play football.

We can also put this in another way and say "When it is *not* raining, play football. Otherwise, play video games indoors."


Instead of sensing whether it is raining, we now sense whether it is ***not*** raining.

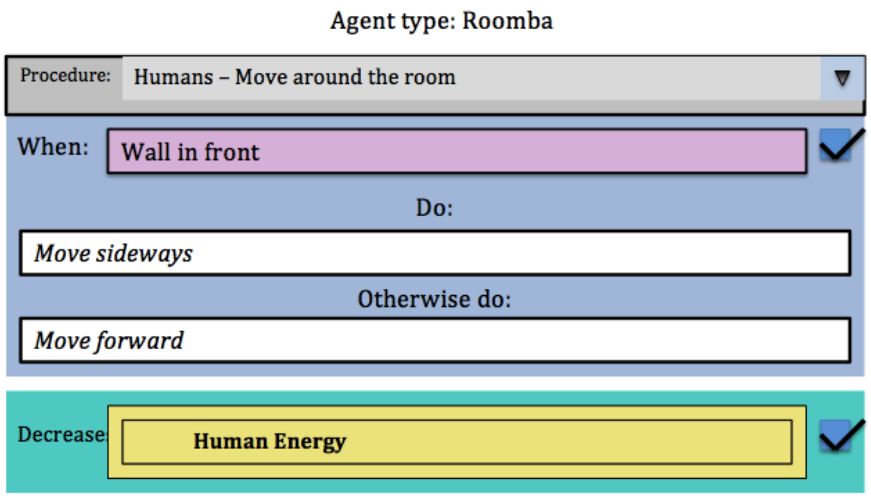
When: *It is not raining outside* ☑

Do: *Play football*

Otherwise do: *Play video games indoors*

When it is ***not*** raining outside, play football. Otherwise, play video games indoors.

Similarly, think of the scenario described on the previous page where Roomba cleans the floor when humans switch Roomba on. We describe the "Clean" procedure for Roomba using the "When…Do…Otherwise do…" block as shown below.


Agent type: Roomba

Procedure: Humans – Move around the room ▼

When: Wall in front ✓

Do: *Move sideways*

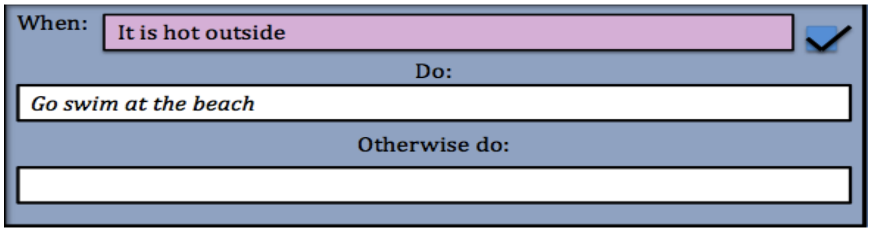
Otherwise do: *Move forward*

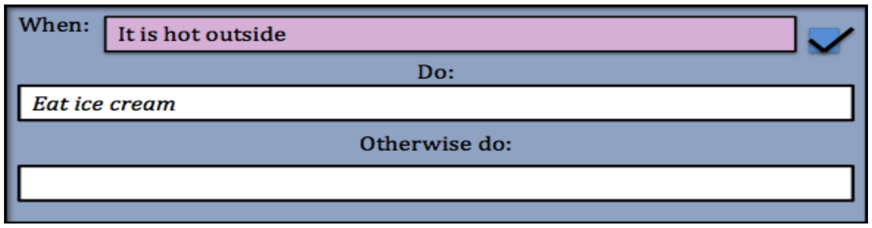
Decrease: **Human Energy** ✓

Sometimes, you may need multiple "When…Do…Otherwise do" blocks to represent more complex sense-act processes.

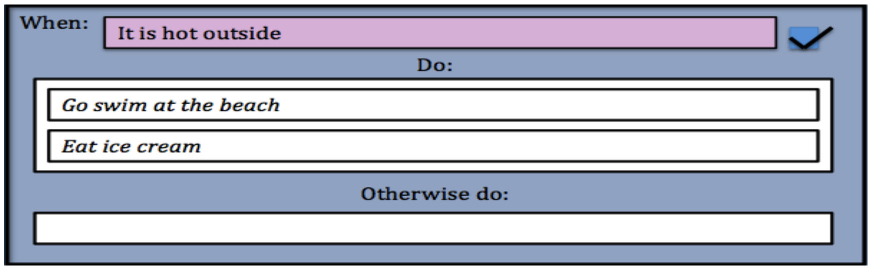
Representing multiple actions under one condition

Sometimes, when one condition is true, we act in multiple ways. For example, when it's hot outside, we may decide to go swimming at the beach and eat ice cream. One way to represent these two actions is how we did in the previous page.

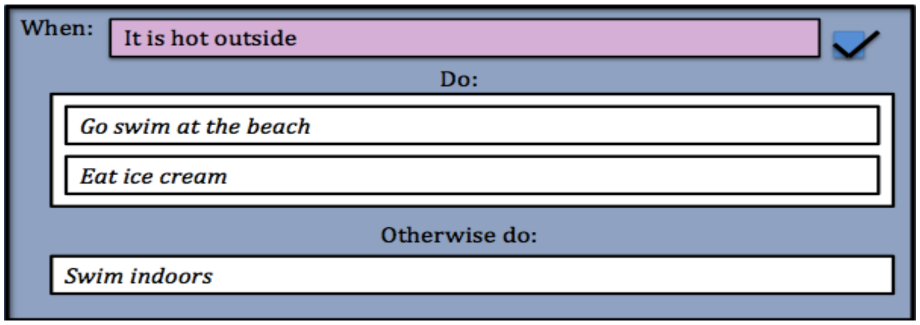




But we can also represent two actions under one condition using only one "When…Do…Otherwise Do" block instead of two. To do this, we simply need to drag and drop multiple actions under "Do:" and it will look like the block below. This block tells us that when the condition, "It is hot outside," is satisfied, we go swim at the beach and we also eat ice cream.



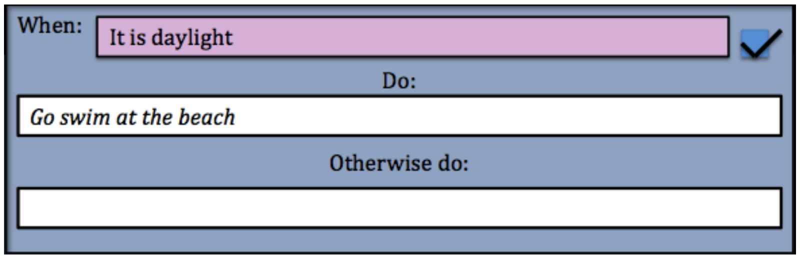
We can also decide to do something else when the condition is not satisfied. For example, if it *not* hot outside, we may decide to swim indoors instead. See the block below. It says that when it is hot outside, we go swim at the beach and eat ice cream; but if it not hot outside, we swim indoors.

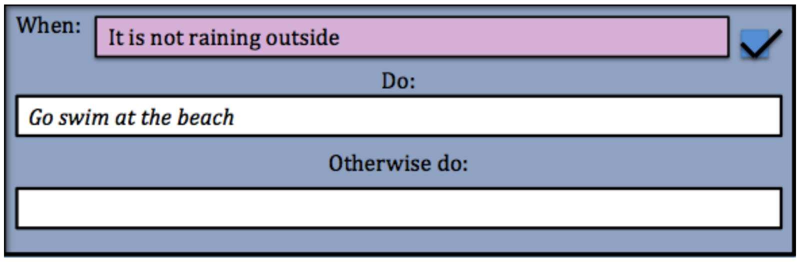


Representing actions which happen when multiple conditions are true

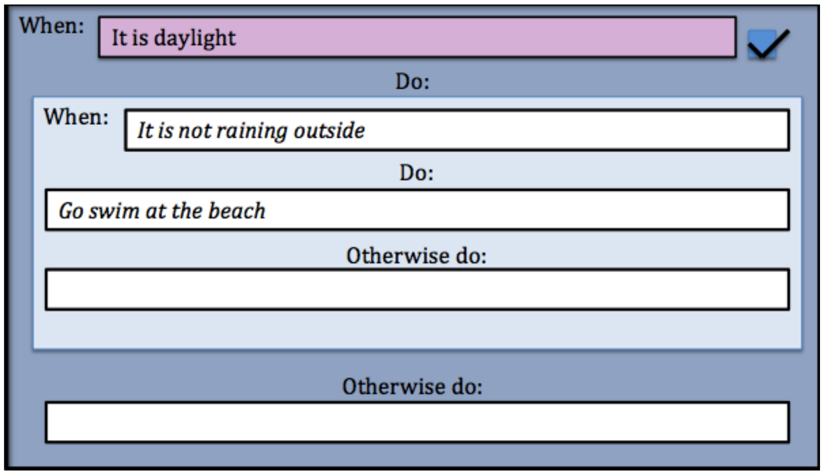
So far, we have seen that when one condition is true, we act in one way or in multiple ways; and when that condition is not true, we act in a completely different way. But what if multiple conditions need to be true before we can act? For example, if you want to go swim at the beach, these two conditions need to be satisfied: (1) It is daylight and (2) It is not raining.

You could try representing the scenario using two sets of "When…Do…Otherwise do" blocks as shown below, but do you think they correctly represent the scenario in which multiple conditions need to be satisfied before an action can happen?





Actually, the two "When…Do…Otherwise do" blocks above are incorrect. Why? Well, if it is daylight outside but it is raining, you cannot go swim at the beach. Similarly, if it is not daylight outside, you cannot go swim at the beach, whether or not it is raining. Thus, both of the conditions above need to be true at the same time. In other words, when it is daylight, we need to make sure it is not raining outside, and if it is not raining outside, then we go swim at the beach. So how can we represent these multiple conditions in "When…Do…Otherwise do" blocks correctly? Look at the nested blocks below.

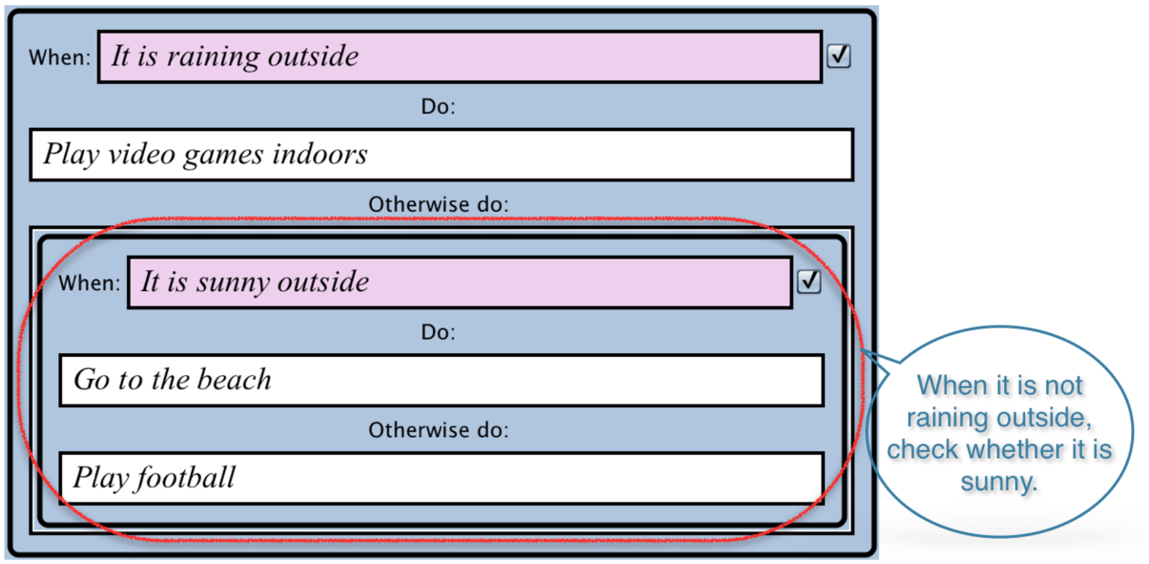


What we have done here is dragged one "When…Do…Otherwise do" block into the "Do" segment of another "When…Do…Otherwise do" block. We will need to do this every time we need to represent actions which take place when multiple conditions are true at the same time.

Representing complex "Sense-Act" processes

In the previous example, we said "When it is raining outside, play video games indoors. Otherwise, play football." So we play football whenever it is not raining, that is to say, when it is cloudy or sunny.

But what if we only want to play football only when it is cloudy? What if we want to go to the beach when it is sunny? There are many situations like this, where we want to sense more than 1 condition. Is there a way to express these complex "sense-acts" with the "When … Do … Otherwise do …" block? Yes, like this:

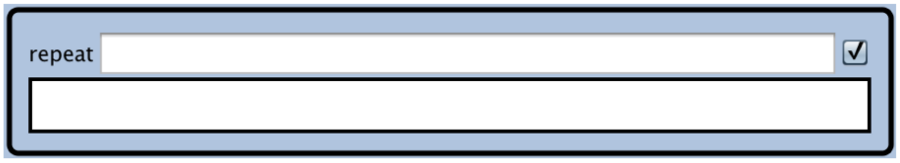


When it is raining outside, play video games indoors. When it is sunny outside and not raining, go to the beach. When it is neither raining nor sunny outside, play football.

In this "When … Do … Otherwise do …" block, we first check whether it is raining outside. When it is raining outside, we play video games indoors. When it is not raining outside, we check whether it is sunny. When it is sunny, we go to the beach. Otherwise, when it is neither raining nor sunny outside, we go play football.

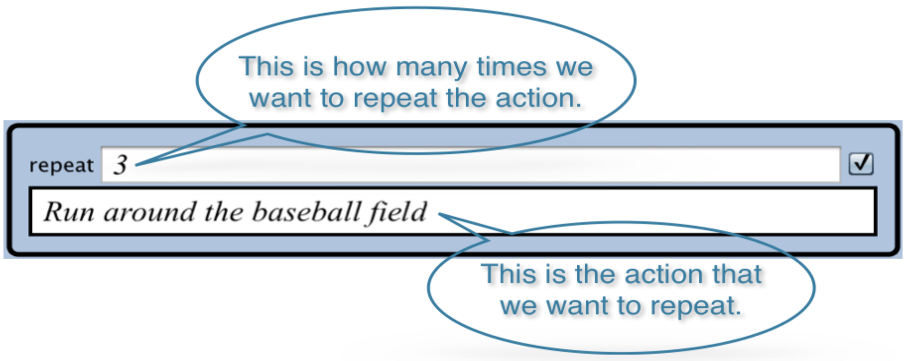
The "Repeat" command

We use the "Repeat" block to perform an action multiple times repeatedly.



An empty "Repeat" block

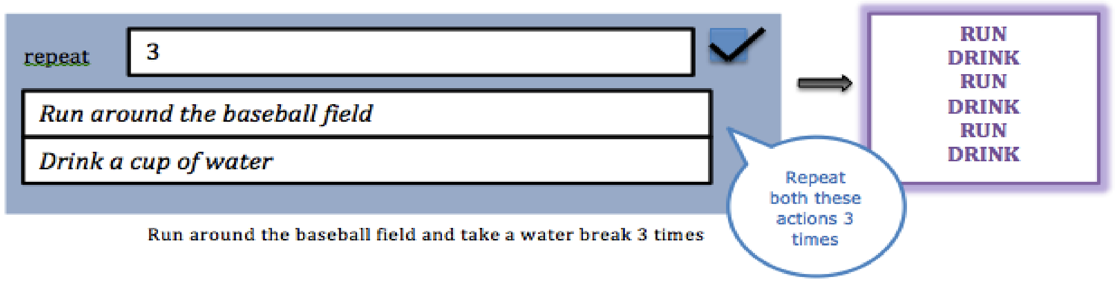
For example, your fitness coach may tell you to run around the baseball field 3 times. Here's how we would express this:


This is how many times we want to repeat the action.
repeat 3
Run around the baseball field
This is the action that we want to repeat.

Run around the baseball field 3 times

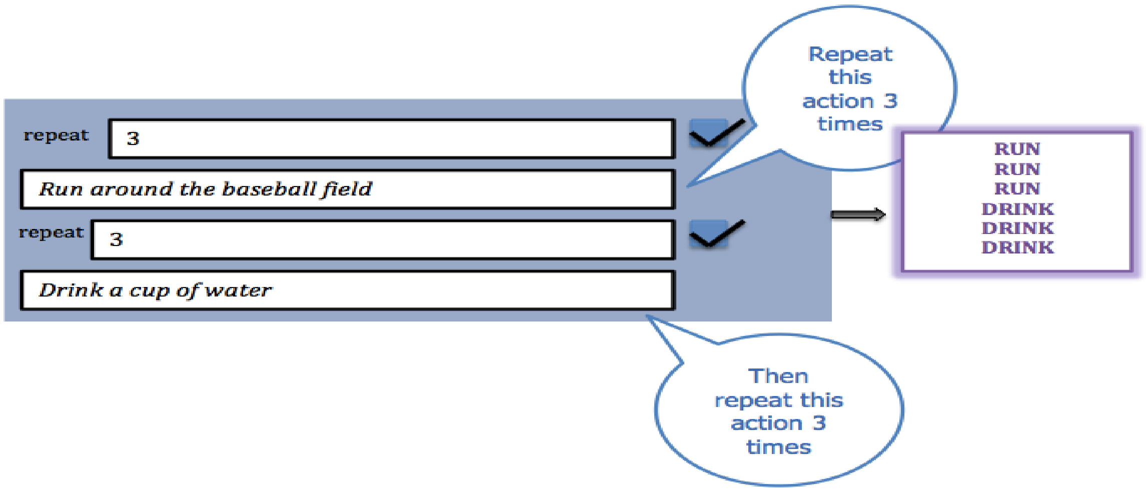
As we can see from this example, first we put in the number of times that we want to repeat the action, and then we put in the action that we want to repeat.

We can also repeat multiple actions in a "Repeat" block. For example, if you had to run around the baseball field and then take a water break before running another round and had to do this exercise 3 times, you could express it as follows:


repeat 3
Run around the baseball field
Drink a cup of water
Repeat both these actions 3 times
RUN
DRINK
RUN
DRINK
RUN
DRINK

Run around the baseball field and take a water break 3 times

Can you predict what would happen if we used the Repeat command in this way instead:


Repeat this action 3 times
repeat 3
Run around the baseball field
repeat 3
Drink a cup of water
RUN
RUN
RUN
DRINK
DRINK
DRINK
Then repeat this action 3 times

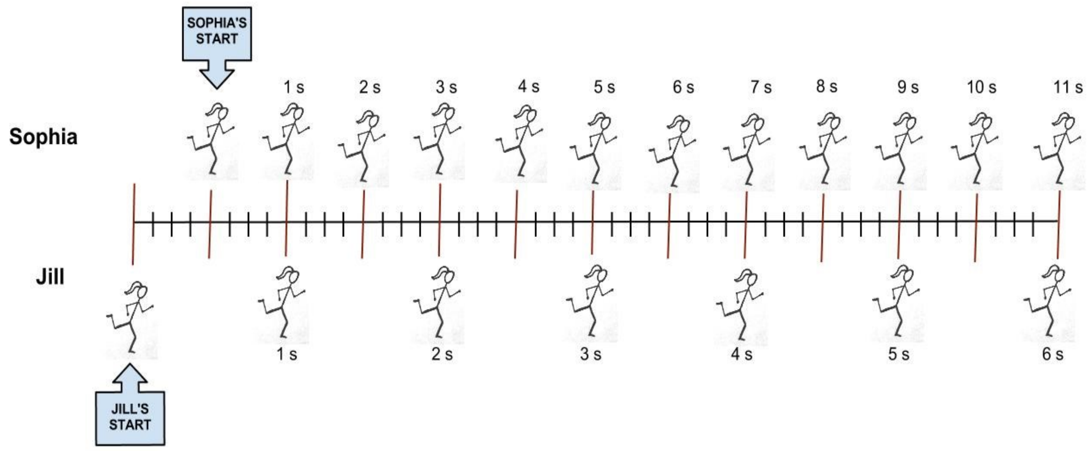
APPENDIX B

Paper-based assessment artifacts used with CTSiM v2

B.1 Pre/post tests

B.1.1 Kinematics pre/post test questions

1. Sophia and Jill start running at the same time and run in the same direction. In the diagram below, Sophia's and Jill's starting positions are marked. Their positions at every second after they start running are also shown. Each small hash mark represents one foot, and "s" means *seconds*.



a. Who is running faster, Sophia or Jill? Be sure to explain your answer using the numbers in the diagram.

b. Do Sophia and Jill accelerate as they run?

Sophia accelerates: (A) YES (B) No

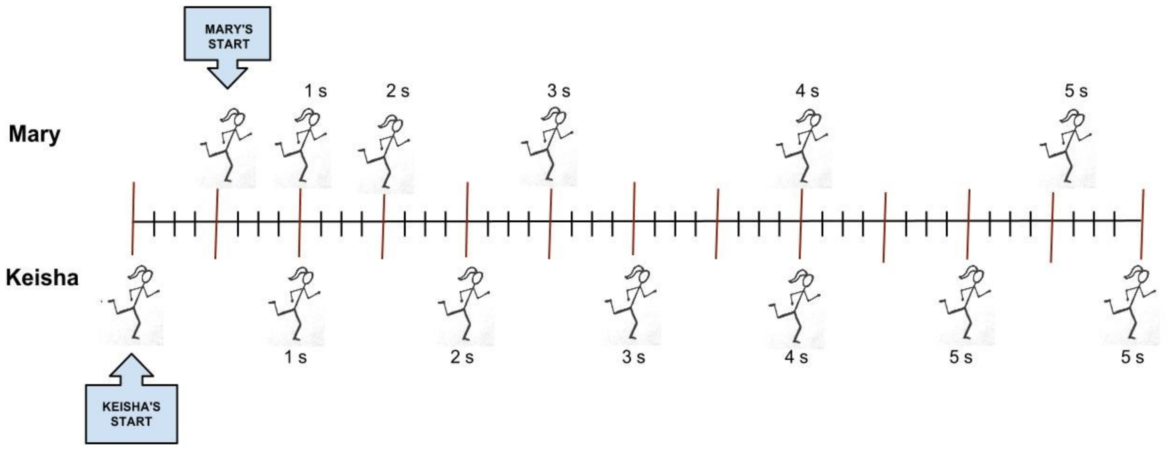
Jill accelerates: (A) YES (B) No

c. Whose acceleration is greater?

(A) Sophia (B) Jill (C) Both accelerate at same rate (D) Neither accelerate

2. Mary and Keisha start running at the same time from left to right as shown in the diagram below. Their positions as they run are marked and numbered at every second. Each small hash mark in the picture represents 1 foot, and "s" means *seconds*.

Do Mary and Keisha ever have the same speed?



(A) No, they do not.

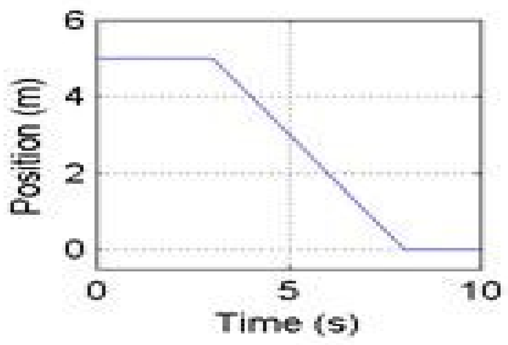
(B) Yes, at time = 1s.

(C) Yes, at time = 4s.

(D) Yes, at time = 1s and time = 4s.

(E) Yes, during the time period from 2s to 3s.

3. The graph below tells us how a ball's position changed with time. Which of the following best describes the ball's motion?



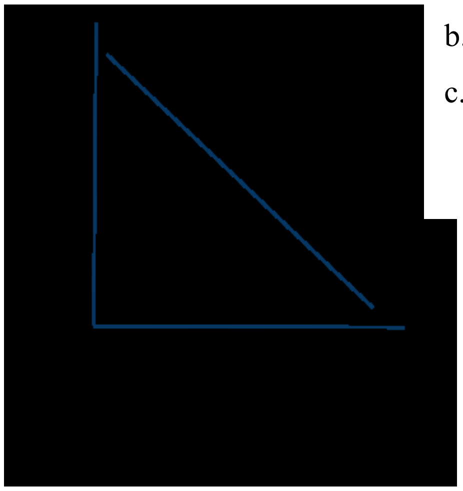
a. The ball moves along a flat surface. Then it moves forward down a hill, and then finally stops.

b. The ball is moving at constant velocity. Then it slows down and stops.

c. The ball doesn't move at first. Then it moves backwards and then finally stops.

d. The ball moves along a flat area, moves backwards down a hill and then it keeps moving.

4. The speed-time graph below describes the motion of a moving object. Use the graph to answer some questions about the motion of the object.



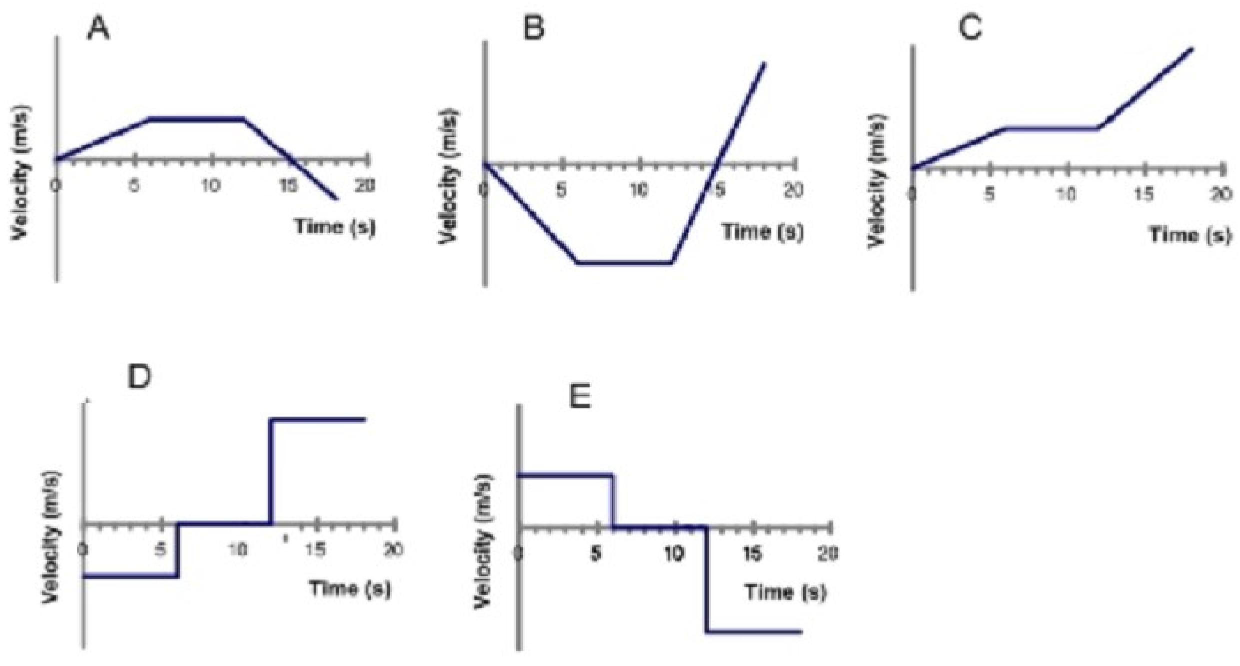
a. What happens to the speed of the object over time?

b. What happens to the acceleration of the object over time?

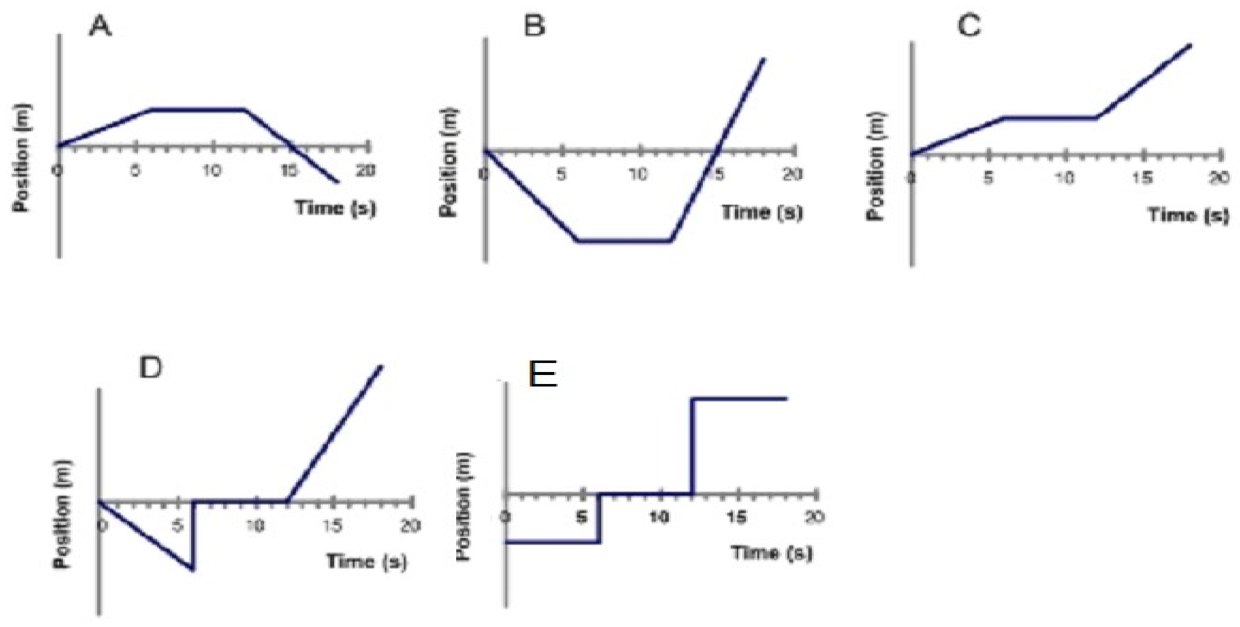
c. What happens to the distance travelled by the object per time unit?

5. A man starts at the origin (position =0 at origin) and walks backwards slowly at a constant speed for 6 seconds. Then he stands still for 6 seconds, and then walks forward at a constant speed that is twice as fast as before for 6 seconds.

i. Circle the **velocity time graph** which best depicts the man's motion.

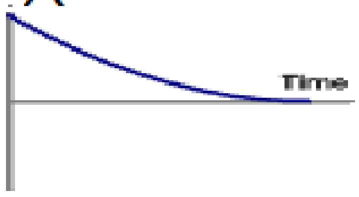


ii. Circle the ***position-time* graph** which best depicts the man's motion

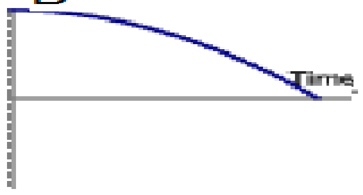


6. A car is moving forward and applying its brakes. Circle the **position-time graph** which best depicts this motion.

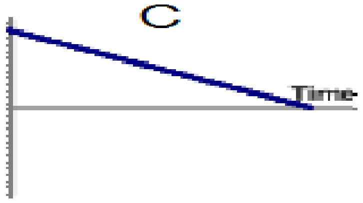
A



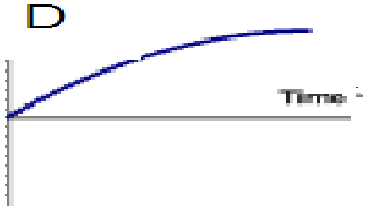
B



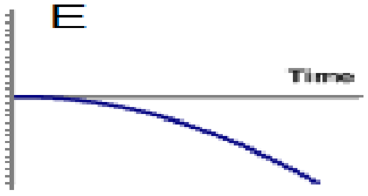
C



D



E



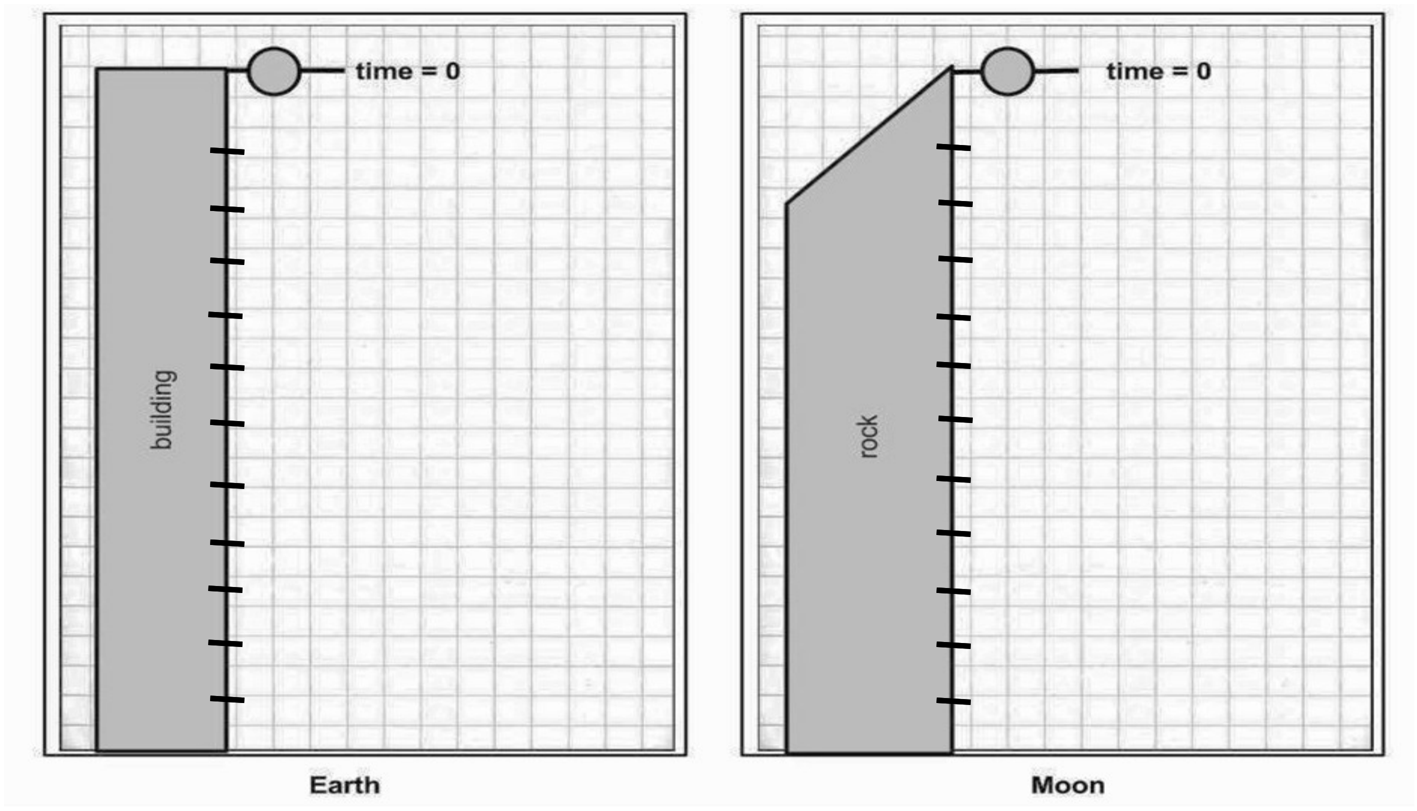
7. Consider a traffic scenario. A car arrives at a traffic-light at time=0 second, and sits at the red light for 5 seconds. After 5 seconds, when the light turns green, the car starts increasing its speed until it reaches a velocity of 10m/s at time = 12 seconds. After that, the car maintains its speed at 10m/s. Can you predict what the position-time and velocity-time graphs will look like for this scenario?

Velocity-time (x-axis=time, y-axis=velocity)	**Position-time** (x-axis=time, y-axis=position)

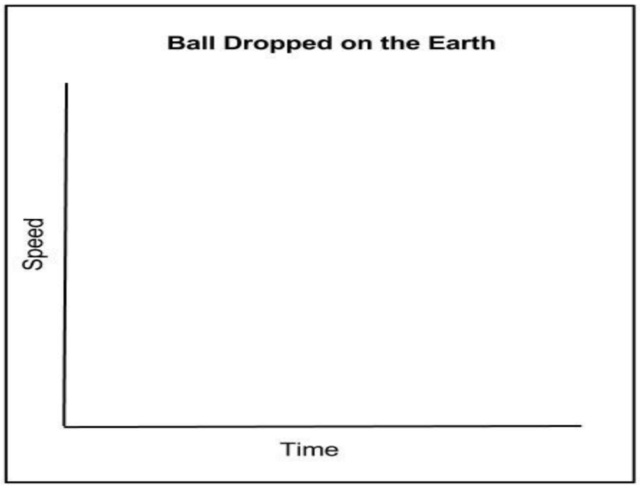
8. A ball is dropped from the top of a building on the Earth and takes 5 seconds to reach the ground. A second ball, identical to the first one, is dropped from the top of a tall rock on the surface of the moon that is of same height as the building. It takes more than 5 seconds for the second ball to reach the moon's surface.

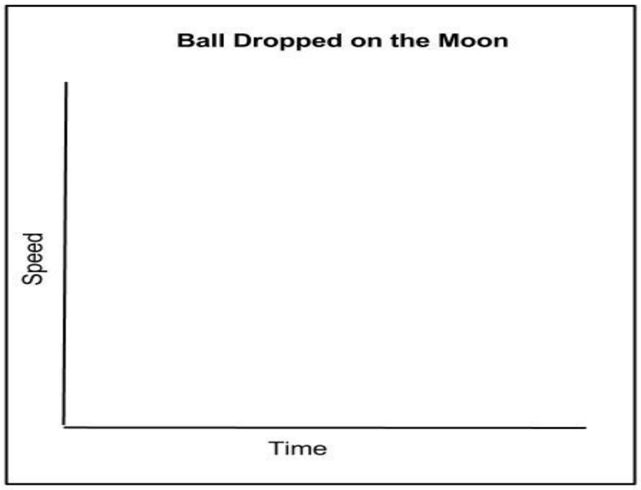
Predict the position of the ball after 1, 2, and 3 seconds on the Earth and also on the moon by drawing on the figures below.

The acceleration due to gravity on the ball is greater on the Earth than it is on the moon, **so the ball dropped on Earth will speed up faster than the ball dropped on the moon.**



9. Draw graphs to show how the speeds of the balls in Question 8 vary with time.





B.1.2 Ecology pre/post test questions

You are provided with a fish tank which contains the following:

Living species	**Chemicals**
1. Goldfish	5. Oxygen (O_2)
2. Duckweed	6. Carbon-dioxide (CO_2)
3. Nitrosomonas bacteria	7. Ammonia
4. Nitrobacter bacteria	8. Nitrites
	9. Nitrates

Now, answer questions 1-5 with respect to this fish tank.

1. For each of the following species in the fish tank, mention which of 1-9 it directly needs to stay alive. The first species, goldfish, has been filled in as an example.

Example Goldfish __2, 5_____

Duckweed	__________
Nitrosomonas bacteria	__________
Nitrobacter bacteria	__________

2. Explain the roles of the bacteria in the fish tank.

a. Give **2 reasons** why the **Nitrosomonas** bacteria are important in the fish tank.

b. Give **2 reasons** why the **Nitrobacter bacteria** are important in the fish tank.

3. Your fish tank is currently healthy and in a stable state. Now, you decide to remove all traces of Nitrobacter bacteria from your fish tank. Would this affect

a) Duckweed: ☐ Yes ☐ No

If you answered Yes, explain how duckweed would be affected:

b) Goldfish: ☐es No ☐

If you answered Yes, explain how goldfish would be affected:

c) Nitrosomonas bacteria: Yes ☐ No ☐

If you answered Yes, explain how Nitrosomonas bacteria would be affected:

4. Imagine a fish tank with only one fish, some duckweed and some bacteria. The fish tank is in balance and all the species have sufficient food to consume and enough oxygen and carbon-dioxide to breathe at all times.

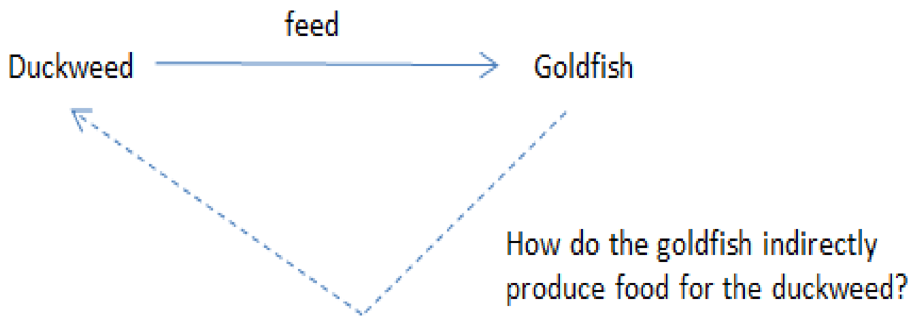
a. If you add a second fish to this fish tank, what do you think will happen? Will it disturb the balance? Will the living species in the fish tank still be able to survive? Explain your answer.

b. Now, think of what will happen to the fish tank if you add 20 more fish. Explain what will happen to the different living species and chemicals in the fish tank

5. Two organisms are said to have a *symbiotic relationship* when they mutually benefit each other. In the fish tank provided to you, the goldfish and the duckweed have a symbiotic relationship.

a. How do the goldfish and the duckweed benefit from each other's **respiration processes**?

b. **Other than the respiration process**, there is another way in which the goldfish and duckweed benefit each other. As shown in the figure below, the duckweed directly acts as a source of food for the goldfish that helps them gain energy, but the goldfish also indirectly provides nutrients for the duckweed. **Describe how the goldfish indirectly produces nutrients for the duckweed helping the duckweed gain energy. Note that this answer may require more than one step.**


feed
Duckweed
Goldfish
How do the goldfish indirectly produce food for the duckweed?

B.1.3 CT pre/post test questions

1. Emma writes code which says

Repeat 2

   [**Do a math problem**]

Write an essay

while,

John writes code which says

```
Repeat 2
  [Do a math problem
   Write an essay]
```

Which of the following statements is correct?

a. Both Emma's and John's code say: Do 2 Math problems and then write 2 essays.
b. Emma's code says: Do 2 Math problems and then write one essay, while John's code says: Do 2 Math problems and then write 2 essays
c. Emma's code says: Do 2 Math problems and then write an essay, while John's code says: Do a Math problem, write an essay, then Do a 2nd Math problem and then write a 2nd essay
d. Emma's code says: Do a Math problem, then write an essay, and then Do a 2nd Math problem, while John's code says: Do a Math problem, then write an essay, then Do a 2nd Math problem and then write a 2nd essay

2. Consider the following program

```
If (quiz-score is equal to 10)
    Then: Get the 'You're a pro' sticker
    Else: _____
If (quiz-score is greater than 7)
    Then: Get the 'Good job' sticker
    Else: _____
```

Bill gets a score of 9 on the quiz while Janet scores 10 points on the quiz. What stickers should Bill and Janet receive based on the above program?

a. Bill: 'Good job' sticker; Janet: 'You're a pro' sticker
b. Bill: 'Good job' sticker; Janet: 'Good job' sticker
c. Bill: 'Good job' and 'You're a pro' stickers; Janet: Good job' and 'You're a pro' stickers
d. Bill: 'Good job' sticker; Janet: Good job' and 'You're a pro' stickers

3. Consider the following program

```
If (quiz-score is greater than 7)
    Then: If (quiz-score is equal to 10)
            Then: Get the 'You're a pro' sticker
            Else: Get the 'Good job' sticker
```

```
        Else: Get the 'Try harder' sticker
```

Bill gets a score of 9 on the quiz while Janet scores 10 points and Kim scores 5 points on the quiz. What stickers should they receive?

a. Bill: ____

b. Janet:____

c. Kim:____

4. Consider the following program:

```
    If (time is after 6 pm)
        Then: Work on science project
        Else: If (time is after 3 pm)
                Then: Play with friends
                Else:_________
```

Jonah is in California and it is 4 pm, while Betty is in New York and it is 7 pm. What are Jonah and Betty doing based on the given code above?

a. Jonah: work on science project; Betty: play with friends.

b. Jonah: play with friends; Betty: work on science project.

c. Jonah: work on science project and play with friends; Betty: play with friends.

d. Jonah: work on science project; Betty: work on science project and play with friends.

5. You are training a robot to avoid obstacles as it moves. To make things more interesting you tell the robot to turn right to go around the obstacle if the color of the obstacle is red. If the obstacle is of any color other than red, the robot should turn left to go around the obstacle. How will you program your robot to follow these instructions using a **When-Do-Otherwise do** structure?

When: ____

Do: ____

____

Otherwise do: ____

____

6. Imagine you have established a colony on the moon, and have robots helping you with your tasks. Today you need to program your robot to go test an instrument that is 5 miles away. Your robot can only travel 1 mile on a fully charged battery. Fortunately, you have a charging station every mile long the way. Using the loop structure (the **Repeat** command), write a program to command your robot to successfully reach the instrument so it can conduct the test. Assume your robot is fully charged when it starts its mission.

7. Do you know how ant colonies gather food? Each ant has two primary tasks: looking for food and returning to the nest with the food. When an ant finds a piece of food, it returns to its nest with the food and releases a chemical as it moves. Ant nests have a specific smell which helps ants find their way back to the nest. When ants look for food, they sniff the scent of the chemical and follow the scent toward the food. As more ants carry food to the nest, the chemical trail becomes well defined.

Can you conceptualize how to describe the ant colony using an agent based framework?

<u>Agent type(s) and properties for each agent:</u>

Environment element(s) and properties for each environment element:

Agent behavior(s):

1. Behavior name:

Sensed properties:

Acted on properties:

2. Behavior name:

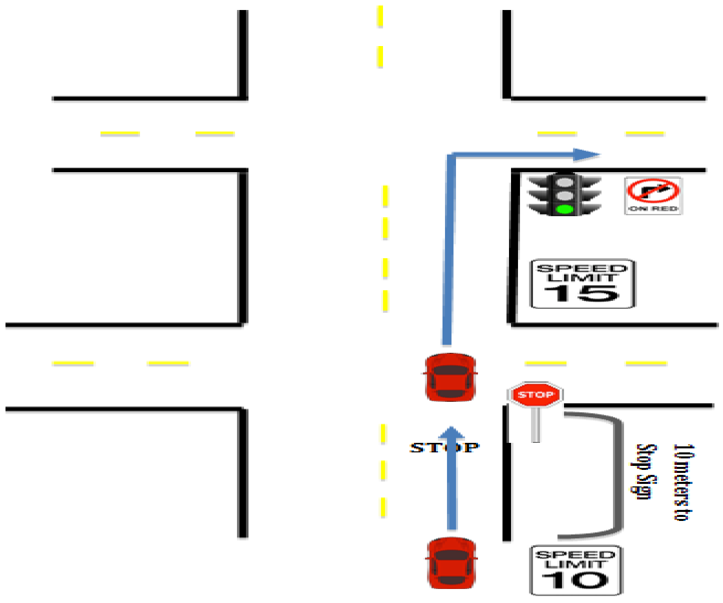
Sensed properties:

Acted on properties:

8. You are given the task of modeling the motion of a car shown in the figure below and how its speed varies on a given path.

The motion of the car can be broken down into 5 different segments. For each segment, specify what things the car will need to sense and what things it will change or act on. Then write a small program to describe the motion in each segment using some or all of the constructs/building blocks provided below. **Do NOT create your own constructs.**

The first two segments have been filled in as an example for you.



Do **NOT** write in the blanks inside the boxes. Use the constructs in your program.

Action constructs:

Set speed = _______ meters/second

Forward ________ meters

Forward at increasing speed till speed reaches _________ mph

Forward at decreasing speed till speed reaches _________ mph

Turn right by ______ degrees

Turn left by _______ degrees

Pause for _______ seconds

Sensing condition constructs:

Stop-sign-visible?

Traffic-light-is-visible?

Car properties:

Speed

Conditional constructs:

When ______________

Do: ________________

Otherwise do: ________

Segment 1: The car initially has cruise control on and is travelling in a straight line at a constant speed of 10 meters/second.

Sense: ______none____

Act on: ____car-location____

Program: _________Set speed = *10* meters/second________________________

____________________Forward *speed* meters__________________________

____

Segment 2: The driver sees a STOP sign at about 10 meters before the stop sign, and begins slowing down the car to come to a stop. Then, she pauses for 5 seconds at the Stop sign.

Sense: ______stop-sign-visibility____

Act on: ____car-location, car-speed______________________________________

Program: ___When: Stop-sign-visible?_________________________________

______________Do: Forward at decreasing speed till speed reaches 0 mph______

__________________Pause for 5 seconds_________________________________

______________Otherwise do:____

Segment 3: The driver then picks up speed to reach the speed limit of 15 meters/second for that stretch of the road.

Sense:

Act on:

Program:

Segment 4: The car needs to turn right at the next intersection. However, the rules say that the car must come to a complete halt at the intersection with traffic lights before it can turn right.

Sense:

Act on:

Program:

<u>Segment 5</u>: There is a No-turn-on-red sign at the intersection. So, the car waits for the traffic signal at the intersection to turn green in order to make a right turn.

Sense:

Act on:

Program:

B.2 Modeling skill transfer test

1. In the following paragraphs, you will read about an ecosystem containing wolf, sheep and grass. You are given the task of modeling this ecosystem and the relations between its species based on what you read. Remember, modeling involves first identifying the agents and environment elements which make up the ecosystem, followed by describing each agent behavior using a sense-act framework and writing a program for each behavior.

First of all, let us read carefully about the ecosystem.

The wolf-sheep ecosystem consists of wolves, sheep, and grass where the wolves prey on sheep to gain energy and the sheep eat grass to gain energy. The wolves prey on sheep whenever they are hungry and sheep are available, and gain 5 units of energy every time they consume a sheep. The sheep, on the other hand, can eat grass whenever they are hungry since grass is always available, and they gain 2 units of energy every time they eat grass.

Other than energy from their food, the wolves and sheep need oxygen to stay alive. They die when they run out of energy or when there is no oxygen present in the atmosphere. Both the animals breathe in oxygen and breathe out carbon dioxide. The grass on the other hand breathes in carbon dioxide and breathes out oxygen. The grass withers and dies if there is no carbon dioxide in the atmosphere.

Wolves and sheep wander randomly in their landscape. Each step they take costs them 1 unit of energy. When they run out of energy they die. Also, each wolf or sheep can reproduce if it has sufficient energy and thus help their populations to continue. Reproduction causes energy to decrease by 4 units for wolves, and by 3 units in case of the sheep population.

(a) Agent type(s) in the ecosystem:

1. ____

2. ____

3. ____

(b) Environment element(s) in the ecosystem:

1. ____

2. ____

3.____

(c) Identify agent behaviors, model them using a sense-act framework, and use some or all of the constructs provided in the boxes below to write programs describing the behaviors.

Action constructs:

Wander

Eat ____________

Die

Create new _______________

Set current energy = previous energy + ___________

Set current energy = previous energy - ___________

Increase

Sensing condition constructs:

The wolf is hungry?

The sheep is hungry?

There are sheep available here?

Some _________ left?

No ________ left?

Conditional constructs:

When _____________

Do: _______________

Otherwise do: ________

Entities:

Sheep Wolves Grass

Agent and Environment properties:

Energy Amount of oxygen Amount of carbon dioxide

Agent 1 – Behavior 1:

Agent name - Behavior name: ______________________________________

Sensed properties:

____

Acted on properties:

____

Program:

____

____

____

____

____

Agent 1 – Behavior 2:

Agent name: Behavior name: ____________________________________

Sensed properties:

____

Acted on properties:

____

Program:

____

____

____

____

____

Agent 1 – Behavior 3:

Agent name - Behavior name: ____________________________________

Sensed properties:

____

Acted on properties:

____

Program:

____

____

____

____

____

Agent 1 – Behavior 4:

Agent name - Behavior name: ____

Sensed properties:

____

Acted on properties:

____

Program:

____

____

____

____

____

Agent 2 – Behavior 1:

Agent name - Behavior name: ____

Sensed properties:

Acted on properties:

Program:

<u>Agent 2 – Behavior 2:</u>

Agent name - Behavior name: ____

Sensed properties:

Acted on properties:

Program:

Agent 2 – Behavior 3:

Agent name - Behavior name: ____________________

Sensed properties:

____

Acted on properties:

____

Program:

____

____

____

____

____

Agent 2 – Behavior 4:

Agent name - Behavior name: ____________________

Sensed properties:

____

Acted on properties:

____

Program:

____

____

____

Agent 3 – Behavior 1:

Agent name - Behavior name: ______________________________

Sensed properties:

Acted on properties:

Program:

Agent 3 – Behavior 2:

Agent name - Behavior name: ______________________________

Sensed properties:

Acted on properties:

Program:

Agent 3 – Behavior 3:

Agent name - Behavior name: ____

Sensed properties:

Acted on properties:

Program:

Agent 3 – Behavior 4:

Agent name - Behavior name: ____

Sensed properties:

Acted on properties:

Program: