

**A Framework for Learning and Reuse in
Visual Programming Environments:
Supporting Novice Programmer Development
of Educational Simulations**

Cheryl Denise Seals

Dissertation submitted to the faculty of the
Virginia Polytechnic Institute & State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science and Applications

Mary Beth Rosson, Committee Chair
Professor, Department of Computer Science

John Burton, Professor
Department of Teaching and Learning

John Carroll, Professor
Department of Computer Science

Roger Ehrich, Professor
Department of Computer Science

H. Rex Hartson, Professor
Department of Computer Science

August 12, 2004
Blacksburg, Virginia

Keywords: reuse, visual programming, simulation creation,
educational simulation, minimalism

Copyright 2004, Cheryl Denise Seals

A Framework for Learning and Reuse in Visual Programming Environments: Supporting Novice Programmer Development of Educational Simulations

By

Cheryl Denise Seals

Committee Chair: Mary Beth Rosson

Computer Science and Applications

ABSTRACT

Incorporating computers into daily K-12 classroom teaching promises to benefit student learning, and improve teaching practice substantially. Computer enhanced curricula may enable more teachers to create exploratory and inquiry based lessons, but in most cases supporting software have only been realized as practice tools for specific rote learning skills. Drills do little to help students develop higher-order reasoning and problem-solving skills. With more computers in the classroom, the assumption was that computers would be integrated into curricula with a high usage of educational software, but research suggests that this assumption has not been borne out (Powell & Okey, 1994; Tyack & Cuban, 1995). Our general argument is that systems whose usability characteristics have been designed to meet teachers' needs and that can be easily tailored to meet specific teaching objectives are more likely to be incorporated into everyday teaching practices.

One type of computer-based activity that enables teachers to engage students in exploratory learning is an educational simulation. Many educational software packages that build simulations have limited usability because they have unmodifiable, limited modifiable or difficult-to-modify functionality. Still others are useful, but are too expensive for many schools to afford. These packages fall short of achieving the ultimate goal of providing useful classroom simulation technology – providing teachers with the option of building simulations from scratch or reusing existing simulations by adapting their functionality.

Because teachers have limited time to learn new technology or develop new simulations, this research focused on developing a new framework that would help teachers create easily adaptable and reusable customized educational materials, encouraging them to use these materials to build and extend simulations in collaboration with their students. We began our study by analyzing the currently available tools for visual construction of educational simulations; we used the results of these analyses to develop an alternative environment—SimBuilder. This environment was designed to address the general usability and programming style issues observed in our analysis of other tools. A minimalist self-study tutorial was designed to support rapid start-up and use of the SimBuilder. Through a comparative analysis using a state-of-the-art environment (AgentSheets) that collected a wide range of quantitative and qualitative measures of learning, programming style, usability, motivation, and strategies for code reuse, we determined that SimBuilder offers an improved environment for teachers to construct educational simulations.

DEDICATION

To my grandmother, Mattie, who always encouraged me and watches over me.

To my grandfather, Lenward, Sr., a man of few words, but always in my corner.

To my grandmother Ida, for being a constant in my life and the rock of her family.

To my father, Lenward, Jr., and Cathy, for just being there.

To my mother, Lillie, always wish you were still here.

To my brothers, Lenward III, Roderick, James & Mike
who keep stretching me and reminding me what's important in life.

To my sister Xanthe, just the two of us, we can make it if we try!

Thanks to all my friends and family who made this journey possible.

ACKNOWLEDGEMENTS

I would like to thank the Virginia Tech Department of Computer Science for their support of my graduate study. I would like to thank my advisor Mary Beth Rosson for all her time in helping me to refine my research, and also for her support and guidance. I would like to thank all the members of my committee, John Burton, Rex Hartson, Roger Ehrich, and John Carroll, for their help, assistance and advice. I am also indebted to the Virginia Department of Teaching and Learning for their help with my studies, particularly Paulette Goodman, who recruited the students for my research investigations.

This study explored the effectiveness of the reuse of both the supplied higher-level components and entire simulations. I would like to acknowledge the members of the Visual Programming Languages Group, Mary Beth Rosson, Nathan Hamblen, Helena Mentis, Stephanie Peppard, and the computer science graduate students at Virginia Tech that participated in our exploratory study. I would also like to thank Alexander Repenning, Andri Ioannidou, Jonathan Phillips, and the Center for Life Long Learning at the University of Colorado Boulder for their help and the opportunity to review and critique their software. Also I would like to thank Larry Tesler, president, Stagecast Software and the creator of Stagecast for providing a copy of their software.

None of this work would have been possible without the financial support of the National Science Foundation, which funded the majority of my research. Also, I received financial aid from the Virginia Commonwealth Fellowship, Department of Computer Science, and the Minorities Academic Opportunities Program at Virginia Tech.

I would also like to thank Glenda and Wayne Scales, Wanda Smith, Kimberly Ware and many, many other friends, and family for encouraging me to make a last push to finish my dissertation.

And above all, I thank God for giving me strength, patience and the blessing of this opportunity.

TABLE OF CONTENTS

ABSTRACT.....	ii
DEDICATION.....	iii
ACKNOWLEDGEMENTS.....	iv
List of Tables.....	vii
List of Figures.....	ix
CHAPTER 1. INTRODUCTION.....	1
1.1 EDUCATIONAL SOFTWARE DEVELOPMENT.....	5
1.2 END USER PROGRAMMING.....	7
1.2.1 Brief Overview of End user Programming Approaches.....	9
1.3 SCOPE OF THESIS.....	11
1.4 PROBLEM STATEMENT.....	12
1.4.1 Visual Environments For Constructing Science Simulations.....	13
<i>Graphical Rewrite Rules are difficult to generalize</i>	14
<i>Confining characters and behaviors to grid cells reduces realism</i>	16
1.5 SUMMARY OF CONTRIBUTIONS.....	16
1.6 ORGANIZATION OF DISSERTATION.....	17
CHAPTER 2. RESEARCH FOUNDATIONS.....	18
2.1 VISUAL PROGRAMMING ENVIRONMENTS.....	18
2.1.1 Programming by Demonstration.....	23
2.1.2 Simulations/Construction Kits.....	31
2.2 OBJECT-ORIENTED PROGRAMMING.....	33
2.2.1 Visual Object-Oriented Programming.....	33
2.3 SOFTWARE REUSE.....	35
2.4 THE USE OF SIMULATION TOOLS BY TEACHERS.....	40
2.5 HUMAN COMPUTER INTERFACE DESIGN.....	44
2.6 MINIMALIST INSTRUCTION.....	47
2.7 USABILITY EVALUATION.....	50
2.8 SUMMARY.....	55
CHAPTER 3: RESEARCH OVERVIEW.....	56
3.1 PROBLEM STATEMENT.....	56
3.2 RESEARCH APPROACH.....	56
3.3 RESEARCH QUESTIONS.....	58
CHAPTER 4. PHASE I: REQUIREMENTS ANALYSIS.....	64
4.1 TAXONOMY OF SIMULATION ENVIRONMENTS.....	65
4.2 INFORMAL USABILITY INSPECTIONS.....	67
4.3 ANALYTIC COMPARISON OF AGENTSHEETS AND STAGECAST.....	71
4.3.1 System Overview.....	71
4.3.2 AgentSheets/Stagecast Feature Comparison.....	76
4.3.3 AgentSheets vs. Stagecast: Scenarios and Claims.....	78
4.3.4 Implications and Conclusions.....	83
4.4. LEARNING AND REUSE IN AGENTSHEETS.....	85
4.4.1 Introduction to Study.....	85

4.4.2 AgentSheets & Visual AgenTalk.....	86
4.4.3 Example-Based Learning & Reuse.....	87
4.4.4 Learning Observations.....	91
4.4.5 Reuse Observations.....	94
4.4.6 Discussion.....	98
4.4.7 High Level Summary of Usability Findings.....	101
4.4.8. Implications for New Simulation Environments.....	103
4.5 LEARNING STUDY OF STAGECAST CREATOR.....	105
4.5.1 Introduction to the Stagecast Creator Study.....	105
4.5.2 Stagecast Creator.....	106
4.5.3 Minimalist Instruction for Stagecast Creator.....	106
4.5.4 Procedure.....	107
4.5.5 Observation and Results.....	108
4.5.6 Discussion.....	110
4.6 LONGITUDINAL GUIDED EXPLORATION OF STAGECAST.....	111
4.7 IMPLICATIONS OF REQUIREMENTS ANALYSIS.....	119
CHAPTER 5. SIMBUILDER DEVELOPMENT.....	122
5.1 SYSTEM REQUIREMENTS.....	122
5.2 USE CASE ANALYSIS.....	126
5.3 CHOOSING AN OBJECT-ORIENTED LANGUAGE AND PLATFORM.....	133
5.3.1 The Environment.....	136
5.3.2 A Scripting Example in Squeak.....	145
5.4 SIMBUILDER.....	147
5.4.1 General Environment.....	149
CHAPTER 6. PHASE III: COMPARATIVE EVALUATION.....	155
6.1 INFORMAL USABILITY INSPECTION.....	156
6.2 SCENARIO AND CLAIMS ANALYSIS.....	160
6.2.1 Learning Scenario and Claim.....	161
6.2.2 Reuse Scenario and Claims.....	167
6.2.3 Implications and Conclusions.....	173
6.3 EMPIRICAL COMPARISON OF AGENTSHEETS AND SIMBUILDER.....	175
6.3.1 Experimental Methods.....	175
6.3.1.1 Population and Sample.....	176
6.3.1.2 Apparatus: Hardware and Software.....	176
6.3.1.3 Experimental Design.....	177
6.3.2 Materials.....	179
6.3.2.1 Informed Consent.....	179
6.3.2.2 Minimalist Tutorials.....	179
6.3.3 Procedures.....	189
6.3.4 Experimental Data Collection.....	191
6.3.4.1 Pre-test Questionnaire.....	191
6.3.4.2 Performance Data and User Observations.....	192
6.3.4.3 Post-test Questionnaire.....	193
6.3.4.4 Retrospective Interviews.....	193
6.3.4.5 User Interface Expert Evaluation.....	194

6.3.5 Results and Analysis	195
6.3.5.1 Participant Background.....	196
6.3.5.2 Learning Sessions: Performance Data	200
6.3.5.3 Learning Sessions: Qualitative Observations	203
6.3.5.4 Reuse Sessions: Performance Data.....	205
6.3.5.5 Reuse Sessions: Qualitative Observations	207
6.3.5.6 User Reactions	210
6.3.5.7 User Simulation Quality: Informal Observations	229
6.3.5.8 User Simulation Quality: Expert Ratings	237
6.3.6. Summary of Comparative Evaluation.....	242
6.3.6.1 Learning	242
6.3.6.2 Programming Style	245
6.3.6.3 Reuse.....	246
6.3.6.4 Expressivity.....	247
6.3.6.5 Summary	248
CHAPTER 7. DISCUSSION AND CONCLUSIONS	250
7.1 RESULTS SUMMARY.....	251
7.2 LESSONS LEARNED	253
7.3 CONTRIBUTIONS	255
7.4 FUTURE WORK	256
7.5 CONCLUSION.....	260
REFERENCES	262
Appendix A: Taxonomy of Visual End User Programming Environments for Educational Simulations	273
Appendix B: Evaluation Matrix of 5 Environments	276
Appendix C: Evaluation of Stagecast	278
Appendix D: Institutional Review Board Forms: IRB Approval	280
Appendix E: Institutional Review Board Forms: Informed Consent.....	282
Appendix F: Pre-Questionnaire	286
Appendix G: AgentSheets Learning and Reuse Tutorials	291
Appendix H: SimBuilder Learning and Reuse Tutorials	308
Appendix I: Post-Questionnaire.....	328
Appendix J: Post-Questionnaire Data and Statistical Analysis	334
Appendix K: Retrospective Interview Questions	351
Appendix L: AgentSheets Simulation Creation Timing, Objects and Rules Created	353
Appendix M: SimBuilder Simulation Creation Timing, Objects and Rules Created.....	362
Appendix N: SimBuilder Session Transcripts	365
Appendix O: AgentSheets SimBuilder Artifact Comparison	386
Vitae.....	390

LIST OF TABLES

TABLE 1.1. <i>ABBREVIATED TAXONOMY OF LANGUAGES STUDIED</i>	10
TABLE 4.1. <i>ABBREVIATED TAXONOMY OF LANGUAGES STUDIED</i>	66
TABLE 4.2. <i>INFORMAL USABILITY INSPECTION OF FOUR SYSTEMS (ENVIRONMENT ISSUES)</i>	68
TABLE 4.3. <i>INFORMAL USABILITY INSPECTION OF FOUR SYSTEMS (DRAWING TOOLS)</i>	70
TABLE 4.4. <i>INFORMAL USABILITY INSPECTION OF FOUR SYSTEMS (RULE CREATION)</i>	70
TABLE 4.5. <i>CLAIMS ANALYSIS OF A REUSE SCENARIO: AGENTSHEETS</i>	81
TABLE 4.6. <i>CLAIMS ANALYSIS OF A REUSE SCENARIO: STAGECAST</i>	83
TABLE 4.7. <i>SUMMARY OF LEARNING SESSIONS</i>	91
TABLE 4.8. <i>SUMMARY OF REUSE SESSIONS</i>	94
TABLE 4.9. <i>REUSE STRATEGIES OBSERVED DURING THE FOOD CHAIN PROJECT</i>	96
TABLE 4.10. <i>REUSE DURING OCEAN WORLD PROJECT</i>	97
TABLE 4.11. <i>GENERAL OUTLINE OF TOPICS FOR MIDDLE SCHOOL PHYSICAL SCIENCE</i>	113
TABLE 5.1. <i>BASIC SYSTEM REQUIREMENTS</i>	124
TABLE 5.2. <i>SMALLTALK MOUSE BUTTONS</i>	136
TABLE 6.1. <i>USABILITY INSPECTION: ENVIRONMENT ISSUES</i>	158
TABLE 6.2. <i>INFORMAL USABILITY INSPECTION: DRAWING TOOLS</i>	159
TABLE 6.3. <i>INFORMAL USABILITY INSPECTION: RULE CREATION</i>	160
TABLE 6.4. <i>TASK CONTEXT FOR LEARNING SCENARIO</i>	162
TABLE 6.5. <i>USER INTERACTION SCENARIO DESCRIBING AGENTSHEETS LEARNING ACTIVITY</i>	162
TABLE 6.6. <i>CLAIMS ANALYSIS OF AN EXPLORATION AND LEARNING SCENARIO: AGENTSHEETS</i>	164
TABLE 6.7. <i>USER INTERACTION SCENARIO DESCRIBING SIMBUILDER LEARNING ACTIVITY</i>	165
TABLE 6.8. <i>CLAIMS ANALYSIS OF AN EXPLORATION AND LEARNING SCENARIO: SIMBUILDER</i>	166
TABLE 6.9. <i>TASK CONTEXT FOR REUSE SCENARIO</i>	168
TABLE 6.10. <i>USER INTERACTION SCENARIO DESCRIBING AGENTSHEETS REUSE ACTIVITY</i>	169
TABLE 6.11. <i>CLAIMS ANALYSIS OF A REUSE SCENARIO: AGENTSHEETS</i>	170
TABLE 6.12. <i>USER INTERACTION SCENARIO DESCRIBING SIMBUILDER REUSE ACTIVITY</i>	171
TABLE 6.13. <i>CLAIMS ANALYSIS OF A REUSE SCENARIO: SIMBUILDER</i>	172
TABLE 6.14. <i>DESIGN OF LEARNING EXPERIMENT (WITHIN-SUBJECTS: LEARNING PHASE)</i>	178
TABLE 6.15. <i>DESIGN OF REUSE EXPERIMENT (WITHIN-SUBJECTS: REUSE PHASE)</i>	178
TABLE 6.16. <i>SUMMARY OF LEARNING AND REUSE TUTORIAL ACTIVITIES</i>	188
TABLE 6.17. <i>EXPERIMENTAL INSTRUMENTS AND MEASURES</i>	191
TABLE 6.18. <i>GUIDELINES FOR EXPERT RATINGS OF PARTICIPANT SIMULATIONS</i>	195
TABLE 6.19. <i>PARTICIPANT BACKGROUND DATA</i>	196
TABLE 6.20. <i>LEARNING SESSION TIMES FOR SIMBUILDER VERSUS AGENTSHEETS GROUPS</i>	200
TABLE 6.21. <i>ARTIFACT COMPLEXITY MEASURES FOR SIMBUILDER VS. LEARNING</i>	202
TABLE 6.22. <i>REUSE SESSION TIMES FOR SIMBUILDER VS. AGENTSHEETS</i>	205
TABLE 6.23. <i>ARTIFACT COMPLEXITY MEASURES FOR SIMBUILDER VS. AGENTSHEETS REUSE</i>	206
TABLE 6.24. <i>BI-POLAR RATING SCALES ASSESSING GENERAL USER SATISFACTION</i>	211
TABLE 6.25. <i>LIKERT-STYLE RATING SCALES ASSESSING GENERAL EASE OF USE</i>	212
TABLE 6.26. <i>LIKERT-STYLE RATING SCALES ASSESSING MOTIVATION</i>	212
TABLE 6.27. <i>LIKERT-STYLE RATING SCALES ASSESSING PROGRAMMING STYLE REACTIONS</i>	213
TABLE 6.28. <i>LIKERT-STYLE RATING SCALES CONTRASTING REUSE STYLE REACTIONS</i>	214
TABLE 6.29. <i>EVALUATION RATING GUIDELINES</i>	238
TABLE 6.30. <i>USER SIMULATION QUALITY RATINGS BY EXPERT JUDGES</i>	241

LIST OF FIGURES

FIGURE 1.1 GRAPHICAL REWRITE RULE	14
FIGURE 2.1 SIMCITY4 TERRACINA EXAMPLE.....	21
FIGURE 2.2 LABVIEW FRONT PANEL.....	22
FIGURE 2.3 PYGMALION SCREENSHOT: 6 FACTORIAL DEMO	25
FIGURE 2.4 REHEARSAL WORLD	27
FIGURE 2.5 KIDSIM INTERFACE AND RULE EDITING WINDOWS.....	30
FIGURE 2.6 ACTIVCHEMISTRY	33
FIGURE 2.7 ESCOT	38
FIGURE 2.8 TAPPED IN AT STANFORD RESEARCH INSTITUTE	40
FIGURE 3.1 RESEARCH APPROACH	57
FIGURE 4.1. PHASE I REQUIREMENTS ANALYSIS	64
FIGURE 4.2. AGENTSHEETS BEHAVIOR PALETTE (I)	72
FIGURE 4.3. AGENTSHEETS BEHAVIOR PALETTES (II)	73
FIGURE 4.4. STAGECAST CREATOR FLOWER GARDEN	75
FIGURE 4.5. THE WATER CYCLE MODEL USED FOR LEARNING	88
FIGURE 4.6. REUSE MODELS	89
FIGURE 4.7. THE VOLCANO MODEL PRODUCED BY P3	91
FIGURE 4.8. SAMPLE RULE WITH DIRECTIONAL INDICATORS	92
Figure 4.9. A volcano built in visual layers	93
FIGURE 4.10. AGENTSHEETS INTERFACE	104
Figure 4.11. Stagecast Creator Rule Creation	107
FIGURE 4.12. TEACHER CREATED PHOTOSYNTHESIS	115
FIGURE 4.13. TEACHER CREATED CARBON DIOXIDE EFFECTS ON RAISINS IN GINGER ALE.....	117
FIGURE 5.1. SIMBUILDER DEVELOPMENT	123
FIGURE 5.2. USE CASE DIAGRAM FOR CREATE A SIMULATION	128
Figure 5.3. General Teacher Interaction With the System and Toolkits.....	130
FIGURE 5.4. ADAPTATION OF EXISTING PROJECTS.....	132
FIGURE 5.5. SQUEAK CONFIGURATION FILES	139
FIGURE 5.6. SQUEAK 2.6 USER INTERFACE	140
FIGURE 5.7. SQUEAK RULES EXAMPLE	140
FIGURE 5.8. SQUEAK TEXT MORPHING	140
FIGURE 5.9. MORPHIC MENUS.....	141
FIGURE 5.10. MORPHIC SCRIPTING AREA.....	141
FIGURE 5.11. MORPHIC SCRIPTING CATEGORY PANES.....	142
FIGURE 5.12. SQUEAK 3.4 SYSTEM INTERFACE.....	143
FIGURE 5.13. SQUEAK 3.4 NAVIGATOR FLAP	144
Figure 5.14. Object Catalog and Playfield options	145
Figure 5.15. Using Paint Kit to Create a Car	145
Figure 5.16. Car Halo of Operational Handles	146
Figure 5.17. Behavior Script for Object to move forward	147
FIGURE 5.18. SQUEAK: WATER CYCLE SCRIPTING AREA	149
FIGURE 5.19. SIMBUILDER GRAPHICAL USER INTERFACE.....	150
FIGURE 5.20. DARK CLOUD OBJECT AND HALO	151
FIGURE 5.21. BASIC SCRIPTING CATEGORY.....	151
Figure 5.22. SimBuilder Prototype Interface.....	153
FIGURE 6.1 COMPARATIVE EVALUATIONS OF SIMBUILDER AND AGENTSHEETS	155
FIGURE 6.2. EVALUATOR APPARATUS SETUP.....	177
FIGURE 6.3. AGENTSHEETS LEARNING SESSION: CHANGING BEHAVIOR OF CLOUDS TUTORIAL PAGE	181
FIGURE 6.4. SIMBUILDER LEARNING SESSION: CHANGING BEHAVIOR OF CLOUDS TUTORIAL PAGE ..	182
FIGURE 6.5. SIMBUILDER LEARNING SESSION: CREATING A VOLCANO TUTORIAL PAGE	183
FIGURE 6.6. EXAMPLE-BASED REUSE IN AGENTSHEETS TUTORIAL PAGE AND OZONE SIMULATION	185
FIGURE 6.7. COMPONENT-BASED REUSE IN SIMBUILDER TUTORIAL PAGE AND STARTER SIMULATION	186

FIGURE 6.8. SIMBUILDER MINIMALIST TUTORIAL PAGE: REUSE SESSION	187
FIGURE 6.9. INTERACTION GUIDE.....	189
FIGURE 6.10. AGENTSHEETS VOLCANO	203
FIGURE 6.11. SIMBUILDER VOLCANO.....	203
FIGURE 6.12. PHOTOSYNTHESIS MODEL	209
FIGURE 6.13. SIMBUILDER OCEAN WORLD MODEL.....	209
FIGURE 6.14. LEARNING SESSION ARTIFACTS.....	229
FIGURE 6.15. LEARNING SESSION VOLCANO ARTIFACTS	230
FIGURE 6.16. SIMBUILDER LEARNING SESSION VOLCANO ARTIFACTS.....	231
FIGURE 6.17. SIMBUILDER LEARNING SESSION VOLCANO ARTIFACTS.....	232
FIGURE 6.18. BEST EXAMPLES OF AGENTSHEETS OCEAN REUSE MODELS	233
FIGURE 6.19. WEAK EXAMPLES OF AGENTSHEETS OCEAN: REUSE MODELS.....	233
FIGURE 6.20. BEST EXAMPLE OF AGENTSHEETS PHOTOSYNTHESIS MODEL	234
FIGURE 6.21. WEAK EXAMPLE OF AGENTSHEETS PHOTOSYNTHESIS MODEL.....	234
FIGURE 6.22. BEST EXAMPLES OF SIMBUILDER OCEAN MODELS	235
FIGURE 6.23. WEAK EXAMPLES OF SIMBUILDER OCEAN MODELS	235
FIGURE 6.24. BEST EXAMPLES OF SIMBUILDER PHOTOSYNTHESIS MODELS.....	236
FIGURE 6.25. WEAK EXAMPLE OF SIMBUILDER PHOTOSYNTHESIS MODEL	236
FIGURE 6.26. USER INTERFACE EVALUATION VOLCANO EXAMPLE	240

CHAPTER 1. INTRODUCTION

Teachers traditionally have not used computers to aid students in active learning, even though computers and new technologies hold remarkable promise for improving day-to-day teaching effectiveness. Although established teachers are better versed in the use of technology as a result of their training in both college and professional development courses, they have most often used computers in unimaginative ways (e.g., mainly for drill and practice). Many teachers are now beginning to explore simple end-user programming tools such as word processors, spreadsheets, grading software, educational simulations, or web page editors (Provenzo, Brett & McCloskey, 1999). However, little is known about how teachers could use such tools to enhance their teaching practices.

A group of researchers at Virginia Tech have been working with teachers in local schools to introduce new ways of using computers in their classrooms. One such endeavor was the Learning in Networked Communities (LiNC) project (Carroll et al., 2000). LiNC collaborated for several years with a group of middle and high school science teachers in the participatory design of a Java-based suite of tools to support collaborative science projects (Isenhour, Rosson & Carroll, 1999). During this collaboration, teacher-participants' general computer sophistication increased tremendously. At the time this study was initiated, these teachers had become capable of both envisioning new computer technology applications and generating detailed specifications for the development of those applications (Chin, Rosson & Carroll, 1997; Chin & Rosson, 1998). However, they were still unable to use applications to create more than formatted text or HTML pages, so they remained unable to develop computer-based science activities unassisted.

Teachers are the conduits for the introduction of any new educational software in the classroom. In a series of informal software requirements analysis sessions with teachers, we determined that many teachers who had used educational software had

become frustrated. Most software they had used overlooked or excluded many of the details they normally provided in regular classroom lessons because those programs focused on the needs of a general audience. The teachers indicated they would find software tools more useful if they provided both specialized support for their curriculum and more concrete learning experiences for their students.

Many teachers have turned to an inquiry-based approach to both teaching and learning. Researchers and teachers are striving to infuse proven, effective practices into teaching so that they can ensure they optimize students' opportunities to become self-regulating learners engaged in reflective inquiry (Mzoughi, 2000; Kyza et al., 2002). As a result, software designers have begun creating software that supports this type of instruction. Simulation-based software has been identified as one way to increase both content-specific inquiry-based software in the classroom and the availability of instructional simulations (Kuyper, 1998; Shrader et al., 2000; Bruce & Easley, 2000).

Visual programming languages have recently been identified as promising resources for teachers for both building and using science simulations (Rader, Brand & Lewis, 1997; Lewis, Brand, Cherry & Rader, 1998; Gilmore, Pheasey, Underwood & Underwood, 1995). Depending on students' grade levels, teachers can use such simulations either to conduct demonstrations (e.g., running a volcano simulation with differing arrays of crust elements to see what happens) or to create models to teach concepts involved in changing environments (e.g., adding new kinds of crust elements, or changing their fissure characteristics). In either case, teachers are able to guide and coach their students more effectively if they either build the original models themselves or are able to analyze and refine models along with their students.

Even teachers who experiment with educational simulations, such as those provided by a research team or as "starting" worlds in an environment, may decide the software is a waste of time and money if it offers an example simulation that maps to just a single topic or scenario that they teach. Therefore, developing an environment where teachers can run and create their own simulations would prove more useful. Such an

environment would give teachers software with greater flexibility and reduce the chance that essential subject details would be lost or misunderstood by providing teachers (the content area experts) with an environment or toolkit that enables them to become end-user programmers who create their own software (Lieberman, 2001).

The research conducted for this dissertation focused on middle school physical and earth science classes. An examination of many candidate pieces of software (e.g., LabVIEW, SimCalc) indicated that these systems only support teachers' needs in limited ways. After studying over two dozen pieces of educational simulation software, it became evident that even those few programs that could be modified by the end-user did not allow the creation of the numerous, varied-content classroom simulations that teachers in our focus groups indicated were necessary for software to be valuable. The teachers in one focus group who worked with the software Stagecast Creator became familiar with the environment and created simulations as curricular aids. Our work with this focus group served as the following: a proof of concept that teachers can produce simulations; evidence that teachers wanted to have specialized aids to support their lessons; and reinforcement of our findings of the limitations with graphical rewrite rule programs, where the brittleness of graphical rewrite rules cause an explosion of rules whenever the user tries to create even slight variations on a theme. Although good simulation software sources already exist in some subject areas, no interactive inquiry-based simulations are yet available for science. The problem with most shrink-wrapped software is that it has been created specifically to fill someone else's needs. Teachers (the end-users) need software that is malleable enough to create new lessons, and to be able to use and modify existing simulations to meet more specialized needs. They also need to be able to carry out these construction or modification activities with very little background in technology or programming techniques.

This research studied two visual programming environments in detail and utilized the results of these studies to develop an easier-to-use visual programming environment designed to support the creation of educational simulations by teachers who are novice programmers. This environment is expected to increase the accessibility of simulation

programming to teachers by giving them a tool that will empower them to create, reuse, and modify a range of classroom-specific educational software. By empowering these teachers as developers, it will become possible to provide them with a wider range of more flexible software. Giving teachers the ability to successively realize a pedagogical need for technology, craft a tool to fill that need, and integrate that tool within their curricula will result in an educational revolution.

A key goal of this research was to empower teachers both as authors and as resource developers. Social scientists who have studied problems with the adoption of educational technology have come to the conclusion that for a new technology to have an impact, it must be both adopted by educators and incorporated into their day-to-day teaching practices. Some software companies approach schools with software touted as helping improve test scores, but since school budgets are meager most marketing campaigns are focused on the at-home market rather than the classroom, in many cases teachers are not even aware of the many packages that may be available for their classes. This marketing problem is caused by dwindling school budgets, and with this as a reality companies do not focus their efforts on the classroom. Teachers are rarely consulted in creating educational software; therefore most educational software generally provides very limited support of the teachers' planned in-class activities. Why do teachers fail to see the usability & reusability of educational software? The best technology in the world, offered with the most refined training materials possible, would still fail if teachers did not appreciate both its value and application to their own teaching activities (Powell & Okey, 1994; Tyack & Cuban, 1995). It was clear from the experience of the LiNC Project that teachers each have unique approaches to teaching, and that the courseware they are willing to use must be customizable to fit their pedagogical goals and teaching style (Koenemann, Carroll, Shaffer, Rosson & Abrams, 1998; Dunlap, Neale & Carroll, 2000).

This research investigated educational software systems used to build simulations and created a solution designed to increase the amount of end-user-customizable software for the classroom. This work addressed the particular needs of teachers as end-users and the ways that a visual simulation environment might be made more usable for this target

population. The remainder of this chapter discusses the research problem in more detail, the general approach taken, and the limitations of the research conducted. It culminates in a summary of the contributions of this work and the high-level organization of the dissertation.

1.1 Educational Software Development

Because school districts have limited financial resources, they must use their money wisely to get the most for their dollars. Soloway (1998) points out that there is a lack of money to create new educational software because schools are spending their limited dollars instead on textbooks.

Although schools are not spending money on software, in part because they are still in the process of building a basic computer infrastructure, the biggest reason they spend little on software is that the United States is one of the few industrialized nations with no national educational standard. Instead, educational standards are mandated by states, and textbook manufacturers design books to support those state standards. Textbooks thus contain facts specific to the subject being taught, lesson plans, pertinent usage examples, exercises, and tests—essentially all of the basics needed by teachers for class preparation. School curricula are built around those state-compliant textbooks so that the students can pass state-mandated tests. This creates a sequence: the creation of new standards is followed by the creation of new textbooks to support those new standards and the cycle repeats *ad infinitum*. Creating a multimedia CD that supports a few lessons in a fashion similar to that used in creating such textbooks is simply not cost-effective. Therefore, textbooks provide the greatest return on investment. Textbooks organize courses into linear sequences of modules, which map well to teachers' daily or weekly goals.

Teachers have limited time in which to learn how to work with new educational software, and their time is already stretched thin by teaching, disciplining students,

grading papers, and required after-school activities. Furthermore, the majority of teachers do not possess the technical expertise required to create their own software. Consequently, most education software is created by software development experts, who are not content experts, rather than teachers, who are. The software teachers have available for their use, therefore, often does not meet their individual curricular needs.

One viable method of introducing new software into the classroom is by creating an environment that allows teachers to craft their own special-purpose software (Soloway, 1998). Simulation construction environments are such an option, in that teachers are able to build an active “world” through which their students can view or explore very specific lesson content. The success of such an environment might be improved if it not only supported simulation programming, but also incorporated a mechanism for facilitating the reuse of other simulations or simulation components. Such an approach could speed the development of simulations, as well as increase the likelihood that teachers would build their projects using tested modules instead of having to build all of the content from scratch. Building from tested components should result in improved software.

There is a solution that already exists which is used to improve curricula for the introductory college course: textbook publishers use elaborate mechanisms to improve curricula with techniques such as manipulatives (e.g. remote controls for group quizzes with immediate feedback), software supplied CD-ROM examples and activities that are delivered with textbooks, sample lectures, and online test batteries. However, booksellers have not extended these elaborate mechanisms to K-12 textbooks as in many cases public schools have problems simply affording enough texts for their students. Flexible simulation creation software that supports the reuse of simulations would bridge this gap and provide more specialized curricular-supporting materials for K-12 education.

Software reuse has been touted as a way to improve the method of software creation. The dream of object-oriented programming (OOP) was that OOP would revolutionize the process of writing computer code in the same way as the invention of

the assembly line did for cars (Cox, 1987). Increasingly, professional programmers rely on object-oriented libraries and frameworks to do their work. As a result, one goal of this study was to develop a similar, although smaller, set of resources that end-users like teachers could manipulate to develop software for their individual classroom needs.

Reusing a library of previously tested simulations and simulation components would allow teachers to build new simulations that both facilitated simulation creation and improved the quality of the simulations created. Thus, ensuring the reusability of components became a key factor in the simulation development process so that teachers could routinely manipulate existing materials instead of, as they do now, only manipulating existing materials on rare occasions.

To increase the chances of success in creating reusable materials, this project focused on simulations for the physical and earth sciences. To some extent this provided protection from criticisms that have been leveled against software libraries created for generic, or general, use. For example, Stevens (1992) doubts whether libraries were ever intended to work for everybody, as they tend to be standardized for particular fields. In addition, a domain-specific system would allow simulations and solutions to be expressed in the particular idiom and at the level of abstraction of the problem domain. "Consequently, domain experts themselves can understand, validate, modify and often even develop domain specific language programs ... Enhance productivity, reliability, maintainability and portability ... allow validation and optimization at the domain level ... [and] improve testability" (van Deursen, 2000, p2). The main goal of this research is to provide an educational tool for school districts with limited financial resources, providing teachers with cost-effective access to more specialized educational software simulations.

1.2 End user Programming

Less than one percent of the population currently knows how to program (Smith, Cypher & Spohrer, 1994). To create more useful software, users must learn to understand

their own software requirements by participating in the creation of the software they use. The problem is that most users do not want to know how to program. Humans are *active* by nature; they just want to get their jobs done (Carroll & Rosson 1987).

Many large companies have created end-user programming systems (e.g. customizable word processors, spreadsheet, and presentation applications) that have been very successful. This has not yet happened, however, in the educational software market. This market is small because there is very little extra money available for educational software. However, teachers could benefit greatly from programs that both help them deliver customized instructional materials and provide students with more opportunities for learning.

Some of the earliest work in end-user programming for educational purposes is associated with Seymour Papert and the LOGO language (Papert, 1977). Papert described the design of LOGO as taking the best ideas about computer science programming language design and "child engineering" those ideas (Papert, 1977). Logo is a text-based educational language derived from Lisp that can be used to teach children techniques of both learning and problem solving.

The initial image of Logo is a turtle in the center of the screen. The programmer types in a set of commands (e.g., "Forward 30") to move the turtle. Longer sets of commands can be typed to program more intricate behavior. In the twenty-five years since Logo was developed, there has been a great deal of progress, both in programming language research and in the development of human-computer interfaces. In the area of end-user programming, however, there is still a need for improved usability. Systems must be easy to learn, easy and pleasurable to use, and flexible (Kuyper, 1998).

A key tradeoff affecting the functionality available in end-user programs is the relationship between ease of use and programming power. Some environments are very easy for novices to use, but in most cases these applications have either fixed (e.g., SimCity) or very limited functionality. Not surprisingly, our analysis of systems with

more programming power confirmed that as power and functionality increased (e.g., LabVIEW with hundreds of pre-programmed widgets for setting up instrumentation), the tool's ease of use decreased proportionally. At best, an environment that supports more complex tasks may hide these functions from the novice user, so that only as users gain experience do they change the system's mode to access the more sophisticated functionalities. The most sophisticated systems, which allow the user to access or modify a wealth of functionality (e.g. LabVIEW), tend to be designed for experienced users. They have an extended learning curve (one LabVIEW user reported it took him about six months to become an experienced developer), making them unsuitable for busy and untrained teachers.

There have been numerous studies of simulation programming with children as the target audience, but in general these have not been complemented by investigations into how end-user programming should be introduced to someone who is both mature and skilled, yet a novice programmer (Brand, Rader, Carlone & Lewis, 1998; Gilmore et al., 1995). Thus, a novel contribution of this research is the focus on middle and secondary school teachers. Empowering these teachers in both software development and reuse should greatly enhance both the amount and value of the software available to them. Teachers serve as their own subject matter experts. When provided with useful and usable programming tools during this study, they proved both able and willing to use them successfully to create their own software.

1.2.1 Brief Overview of End user Programming Approaches

During the review of end-user programming systems, four classes of approaches were considered (see Table 1.1): Preferences, Scripting Languages, Macro Recorders, and Programming by Demonstration (PBD) (Cypher et al., 1993). The Preferences approach presents the user with a predefined set of options to choose from in order to customize an application. This approach is quite narrow, has a specific purpose, and thus does not support general programming tasks. Scripting Languages are small, simple programming languages tailored to perform specific tasks and are thus considered less difficult than a standard programming language. However, to master the scripting

process, users still must learn syntax, semantics, and basic computer science concepts—in other words they must "learn to program."

Table 1.1

Four Classes of End-user Programming Approaches

Class of Approach
<p><i>Preferences</i></p> <p>Predefined set of options that users can choose from to customize an application.</p> <ul style="list-style-type: none"> • Defining a style in Word (e.g., changing alignment from left, to center, to right.) • Filling out an online multiple-choice test or form.
<p><i>Scripting Languages</i></p> <p>Small, simple programming languages that are tailored to perform a specific task and are less difficult than a standard programming language.</p> <ul style="list-style-type: none"> • Html (htmlgoodies.com) • JavaScript (sun.com) • HyperCard Script (Cypher et al., 1993) • Tcl/Tk (tcl.tk)
<p><i>Macro Recorders</i></p> <ul style="list-style-type: none"> • Provide users with a way to record and replay a set of actions. • Recording actions in a Basic spreadsheets (MS Excel) • Telecommunications • Word processing software (MS Word, Claris Works)
<p><i>Programming by Demonstration</i> (Cypher et al., 1993)</p> <p>Based on idea of Macro Recorders, but extends to handle generalized situations.</p> <p>User can specify a new task simply by performing it, and the system will infer/create the corresponding code.</p> <ul style="list-style-type: none"> • Pygmalion (Cypher et al., 1993; Glinert, 1990b) • Stagecast, based on Cocoa and Cocoa, was built from KidSim (Cypher & Smith, 1995) • Rehearsal World (Gould & Finzer, 1984) • AgentSheets (Repenning, 1995)

Macro Recorders provide users with a way to record their actions, and are included in many basic spreadsheet, telecommunications, and word processing software. These mechanisms are very literal. A macro replays a sequence of activities. However, because most repetitive activities are actually repetitive at higher levels of abstraction, replaying a set of actions with preconditions that have changed slightly may produce a totally unexpected outcome. Programming by Demonstration (PBD) is similar to the concept of Macro Recorders, but extends that concept to handle generalized situations. The greatest advantage of PBD over conventional programming is that it enables "Programming in the User Interface." Users are able to simply perform actual tasks that they want to express as programs. PBD devices convert their actions into programmed events, and record those events by translating them into executable programs. This method has great advantages for novice programmers because it greatly reduces the cognitive overhead of learning a new programming language, although the ultimate success of these systems is very dependent on the inferences used for generalization (Myers & McDaniel, 2001). PBD meets users at their points of need. It thus seems to be a very useful area of end-user programming for building educational simulations and as a result was the focus of much of the research conducted during this study.

1.3 Scope of Thesis

This research focused generally on analyzing software that is designed to support K-12 education. However, to limit the scope of the analytic and empirical studies, we focused specifically on software for developing simulations by middle school physical and earth science teachers. Physical and earth science is an obvious domain on which to focus, as these problem areas are ripe with opportunities for visual simulations analogous to the many physical experiments and demonstrations that now support the teaching of these subjects. The goal was to provide a software tool that would allow teachers to build educational simulations, with the hope that these creations would both appeal to and educate their students. To accomplish this, software development focused on both the technical aspects of tools for novice programming and the incorporation of human-

computer interaction techniques that would improve usability for novice teacher programmers. This research explored these teachers' identified needs, examined and identified the limitations of existing systems, and developed an alternative design as a result of these analyses. The results of this research will provide the groundwork for future development of simulation languages and environments both useful to and usable by K-12 science teachers.

1.4 Problem Statement

There is a lack of classroom software that both challenges and helps educate youth. Further, the available software is generally developed by individuals who are not content area experts. Teachers generally buy software that is shrink-wrapped, hoping that it will fulfill their needs and be somewhat pertinent to their lesson plans. This means that the software currently being used has been created to be generic and, therefore, lacks the key elements of theory used by teachers in their individual classes that would make it more valuable in the creation of student learning experiences. At the time of this study, there had been no research examining how teachers used simulation creation software in their classroom activities. Several factors that influence this problem are the following:

- Typical teacher background does not support educational simulation creation
- Teachers face cognitive and motivational obstacles of little time and significant cognitive overhead in learning to create new applications from scratch
- Teachers in general are not trained as professional programmers
- The school setting does not encourage the learning of new technologies
- At the time of this study, there had been no research examining how teachers actually use simulation creation software in their classroom activities.

Thus, one major goal of the project was to further explore teacher needs and preferences for building science simulations. A second major goal was to use this analysis of needs to develop a more appropriate and effective tool for simulation construction.

1.4.1 Visual Environments For Constructing Science Simulations

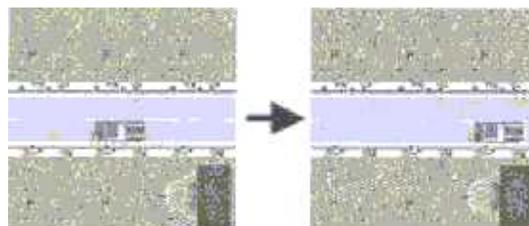
Visual programming is the use of meaningful graphic representations in the process of programming. This study utilizes visual programming, as supported in visual environments, for the constructing of science simulations by novice programmers. One of the most widely used visual environments for constructing science simulations by novices is AgentSheets. AgentSheets is a visual programming tool kit whose simulations operate on a spreadsheet or grid basis. Spreadsheet, grid, table or form based languages refers to languages where the layout of the artificial world (i.e. the simulation microworld) is like a Lotus Notes or Microsoft Excel spreadsheet. Objects negotiate this environment based upon their spatial location, namely their exact cell location. For example, if an object needs to move forward it will move north one grid space, and to move backward it will move south one grid space.

This spreadsheet metaphor was originally introduced to simplify the interpretation of spatial navigation and has proven very successful, being one of the most successful approaches to date in the creation of educational simulations. As an example of the creation of an educational simulation in this environment, consider a water cycle simulation. The model of the water cycle, which is familiar to all earth science teachers, contains several objects: clouds, sun, grass, water (in this case, a lake) which all interact and operate within the model as in a naturally occurring water cycle. These base objects create rain, sunrays, and evaporating mist. To create new objects the user manipulates the drawing tools available in the environment. Once the objects have been created, in order to make it a lively simulation the user must add behaviors to these objects. Rule creation in this environment is base on a toolkit, where the user creates rules using direct manipulation techniques and utilizes rule components from the condition and action palettes. Creating the rule for a cloud moving from place to place, for example, entails creating a rule that would move the cloud from its current position one grid space to the right. A novice user would just create a rule with no *condition* and the *action* tile of “*moving*” and select direction of moving “*to the right*” and their cloud will enact that behavior whenever the simulation is played.

Graphical Rewrite Rules are difficult to generalize

Graphical Rewrite Rule (GRR) systems such as the one employed in AgentSheets and Stagecast Creator are very helpful in getting novice users started quickly. GRRs are interpreted with respect to the object's look, position, and spatial relationship to other objects and are comprised of two situations—a before situation and an after situation. If the predicate condition "situation 1" is satisfied, then the consequent "situation 2" is carried out. This is defined as an action taking place, or a GRR being fired. However, this firing of a GRR can cause considerable problems when the user wishes to change the scenario. For instance, if the user needs to change the position, rotation, or symmetry of an object, all the scenario rules must be individually updated. Specifying all these details can be tedious and there is also a lack of realism in grid-based systems based on a spreadsheet metaphor and the principle of Graphical Rewrite Rules (GRRs).

GRRs present a problem stemming from the implicit nature of rewrite rules and their scalability to more complex problems when they are a form of end-user programming (Repenning, 1995). GRRs describe spatial transformations. Situations can be interpreted with respect to both the objects they contain and the spatial relationships between objects. To create rewrite rules for the scenario of a car moving, for example, the following procedure could be executed. If the predicate condition "situation 1" is satisfied, then the consequent "situation 2" would occur (see Figure 1.1), firing the GRR.



Situation 1

Situation 2

(from Repenning, 1995)

FIGURE 1.1 GRAPHICAL REWRITE RULE

The problem with GRRs is that although a person can infer likely generalizations from a scenario's context, the computer does not have a similar "real world" frame of reference from which to make the same inferences. Thus, the computer must be given explicit visual rules for handling each new situation or context. Such environments require the creation of a computational explosion of rewrite rules to handle what appear to human authors to be very minor situational differences.

Exact visual match requirements are unintuitive

All existing rule-based visual programming systems were developed based on production systems using GRRs. For example, a visual rule specified that if a dog were crouching near a fence, it should jump the fence. In the rule, a crouching dog image was

shown to the left of a fence image . The rule "behavior" showed that the dog image should be moved to the right of the fence image to visually approximate the dog

jumping the fence . Rules interacted based on their direct visual mapping (i.e., appearance) or GRR. When new characters or rules were added, they had to be checked for consistency with the old rules. Often, to ensure consistency, users were required to either update existing rules or create new rules. For example, when the dog's appearance

was changed to standing straight up  from crouching to jump, the visual semantics also changed. As a result, the previously created rule no longer applied to the new visual situation. Another set of rules had to be created to indicate that this visual image of the dog should also (or instead) jump the fence. The exact visual match requirement led to a brittleness in the visual representation that caused problems when there was even a slight change to the simulation or a user wanted to reuse (i.e., incorporate) a rule or character from another simulation.

Exact visual mapping systems that generally rely on macro recorder techniques to create rules thus suffer from the problem of a computational explosion of these rules. Every possible visual situation requires a new set of rules (e.g., if the dog has his tail up) because neither inference nor visual program semantics apply.

Confining characters and behaviors to grid cells reduces realism

Interpretation of rewrite rules limited to syntactic properties makes it laborious for end-users to define non-trivial behavior. Realistically depicted rewrite rules are more difficult to represent than simple depictions, and the kinds of spatial relationships involved are much more difficult to express. In particular, the most difficult problems to address are topology, rotation, symmetry, and the combinatorial explosion of rules required to match different scenarios.

Most systems use a grid or spreadsheet-based notation to provide a mechanism both for specifying the application of a rule in four directions and for sizing objects within the environment. One study goal was to enable a more visually realistic approach that made the sizes of objects and their motions in space less rigid. In the GRR systems reviewed (e.g., AgentSheets in the figure above), the car moved one grid section to the right to drive forward. This proved practical only if the simulation included a straight road surface. In real life, however, this is rarely the case. There are always twists and turns in a road. If the language used to simulate reality can only handle a turn of 4 to 8 dimensions, some realism is lost (see also the alternative technique for addressing limitations of GRRs described by Brownston et al., 1985).

1.5 Summary of Contributions

This research led to a framework for teacher-authoring of visual simulations, thus increasing the likelihood that teachers will craft their own educational materials. It included an analysis of the opportunities and limitations of existing tools for educational simulations and, based upon that information, the development and refinement of a set of tools (SimBuilder and its associated tutorials) that emphasize minimalism, active learning, and component reuse. While assessing the effectiveness of the SimBuilder tool, several activities to facilitate teachers' learning of simulation creation were identified. These activities included supplying them with examples of reuse and modification, utilizing a minimalist tutorial as a guided exploration of their learning experience, and

providing an improved paradigm for programming characters and their behaviors. SimBuilder was also used to provide them with access to its minimally complex environment (e.g., reducing the number of windows that the user has to negotiate). Future work will include study of the customization and creation of simulations by teachers in collaboration with their students using interactive studies in the classroom, as well as whether a virtual support community for teachers will enhance their teaching activities.

1.6 Organization of Dissertation

The dissertation is organized into eight chapters. This introduction is Chapter 1, which provides the general background and motivation. It sets the stage for the proposed work by giving the purpose of the study, the problem being addressed, and the scope and limitations of the research plan. Chapter 2 summarizes the literature in the areas of visual programming, simulation, construction kits, educational software, software reuse, and visual object oriented programming. Chapter 3 elaborates on the goals of this research and introduces specific research questions and hypotheses, along with their assumptions and limitations. Chapter 4 contains the results of our initial requirements analysis. It begins with preliminary and pilot studies and concludes with educational simulation requirements gathered from teachers. Chapter 5 presents the design and development of our refined requirements and the resulting experimental system (SimBuilder). Chapter 6 presents the materials collected during the comparative evaluation of SimBuilder, including a summary of the procedures used for data collection and analysis. The final chapter, Chapter 7, contains the results and conclusions about this work, including summaries of the findings, and recommendations for future work or enhancements.

CHAPTER 2. RESEARCH FOUNDATIONS

As part of this research, pioneering End User Programming (EUP) efforts (e.g., Sketchpad, Pygmalion, SmallStar, Pict, etc.), as well as systems that were visual front ends for textual languages (e.g., Tinkertoy, etc.) were studied. This information included a review of environments that used dataflow, that used actors and agents (e.g., Rehearsal World, ARK, etc.), that were used as User Interface Development Environments (e.g., Garnet, Amulet), and, finally, that supported a constraint-oriented approach (e.g., NoPumpG, etc.). The approaches of the programs reviewed are varied, but their goals in general are the same—to make programming more accessible to novice programmers or end-users. The study focused specifically on those programs designed to support novice end-user programmers in building educational simulations.

In addition to reviewing approaches to visual programming and simulation construction, the concepts of object-oriented programming and design are covered, including a short discussion of code reuse, because these became important considerations in the design of the new environment. The literature describing teachers and their use of technology in the classroom will also be reviewed, as this is the target population and setting for this study. The chapter concludes with a general discussion of the research implications.

2.1 Visual Programming Environments

This research assumes that visual programming approaches would be most effective for novice programmers. Although this is a major assumption, it seems to be consistent with the general trend in user interface design toward visual and interactive environments for end-users, often described under the general terms of direct manipulation, WYSIWYG, or Graphical User Interface. The claim, as applied to novice programmers, is that visual and interactive approaches to programming require less cognitive overhead and learning than textual language-based approaches. Novice users are also assumed to be familiar with direct manipulation paradigms for computer

interaction and are expected to find analogous user interfaces for programming (in this case specifically for simulation construction) more familiar and intuitive as a result.

Users who have become sophisticated users of personal computers and the Internet do not in general wish to become professional programmers, but only to be able to perform their work effectively. Historically, only computer professionals were expected to perform computer programming. Due to the high cost of computer resources, these professionals were expected to work within the limitations of any available language to make applications that would work efficiently with target computers' hardware. Today, the converse is true. Computing power is now much more affordable, and the priority has become to decrease the amount of time users must spend on their computers to accomplish their tasks. This emphasizes the need for new languages and tools designed to support software development by people without specialized computer training—end-user programming (EUP). Visual Programming has been proposed as one way to serve this population. In visual programming, people use graphical representations to specify computational objects and processes.

The general problem with EUP is that programming is hard, so the challenge is to make programming accessible to people whose work can benefit from custom solutions to their own needs. To encourage end-user programming, representations more amenable for human comprehension must be considered. Visual programming is founded on the premise that people prefer pictures to words, because pictures are a powerful and universal form of communication that provide less of a language barrier than natural language (Shu, 1988).

There are some distinctions within and common misconceptions about visual programming. For example, the following are not Visual Programming Languages, but rather are best understood as textual languages with graphical user interface (GUI) builders: VisualWorks, HyperCard, Visual Basic and Visual C++. The programs that are built and executed using these languages actually have a textual representation, being made up of words, characters and other symbols. There is also a distinction between

Visual Environments and Visual Languages. Visual environments focus on supporting visual interaction, while visual languages use visual representations as the primitive building blocks of the languages.

Many applications have addressed the challenge of visual programming by end-users, but most are canned software packages. SimCity is a good example, which offers a work area in which users can assemble and interconnect predefined “city” elements like buildings, roads, cars, and so on. If users want to extend the capabilities of the system, they must acquire the skills of a trained programmer, building their own software and going through an often complex process of integrating their new code within the existing code base.

To illustrate this process, Figure 2.1 shows an image of SimCity created by an end-user sim builder, or mayor, who makes decisions that affect the future of the city. SimCity is a city construction toolkit, where the sim builder begins by choosing a template city. Then within that city they choose what type of housing to add, streets, power supplies, parks, entertainment, malls, etc. The user has to gauge what type of services will be needed to sustain the type of structures that they add to their city. For example, a small city, with a neighborhood of houses, only needs a small power supply, but a large scale city needs power plants, which need a readily available water source, electrical lines and grids, along with highways and trains to bring in materials, in addition to entertainment facilities and parks in close proximity to the living spaces.

In SimCity, a city is successful if it can sustain the number of people that it is built to accommodate and if the population grows. If the population declines drastically and cannot be sustained, there are some parameters that the user can manipulate to help the city flourish. This is both a lot of fun for the user and a mini lesson about city planning. However, although the sim builders can use any of the templates provided, they cannot create their own personalized vision of a city nor change the way in which the dynamics of the city interact.



FIGURE 2.1 SIMCITY4 TERRACINA EXAMPLE
(*simcity.ea.com, 2004*)

Visual programming systems are often based on visual or conceptual metaphors. For example, one class of systems uses visual icons as its language primitives. Users are able to see an icon that represents an object rather than reading a plain text denotation. Several programming systems built to exploit this paradigm of visual representation; particularly those that exploit an object's true visual nature – LabVIEW, Tinkertoy, and Pict – are reviewed here.

LabVIEW is a scientist's construction kit (National Instruments Corp 2001; Schmucker, 1999). Virtual Instruments are built by direct manipulation, with the help of wizards, and a tutorial is included to aid user developers. The system provides three construction palettes, which are used for editing, debugging, and creating. These are the Tools Palette, Controls Palette, and Functions Palette, respectively. Each palette contains icons that are the visual representation of the apparatus necessary to create Virtual Instruments. When these icons are dragged from the palette into a panel to create a

Virtual Instrument, the system creates a corresponding block diagram to indicate the connection of controls and indicators. As LabVIEW is helping industry to become more productive, their claim is that they can also help improve conditions for academic science, as reducing the data gathering time gives researchers and students more time to focus on the concepts and results.

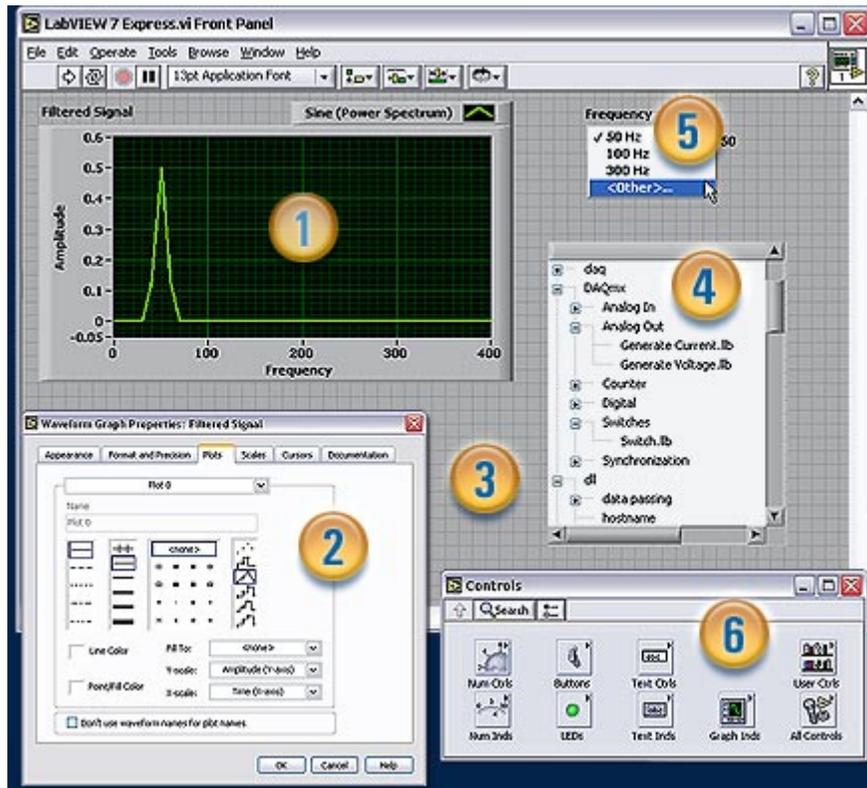


FIGURE 2.2 LABVIEW FRONT PANEL

(National Instruments, www.ni.com, 2004)

Tinkertoy is a graphical visual interface for Lisp, where programs are built with icons and flexible interconnection, rather than typed expressions (Edel, 1986). It provides a consistent framework for interaction where programs are represented graphically. This graphical nature is assumed to improve program readability and understanding (Cypher et al., 1993). The following is an example of a Lisp construct:

(stack block-a block-b) or (defun stack (from to) (move-block from to))

With any textual programming language, users have to make sure they use the correct syntax. A major syntax problem with Lisp is the parentheses. Tinkertoy is

designed to make it easier to understand the flow of control and how to use complicated data structures because its graphic form more closely resembles a list/structure type of internal representation. This drastically simplifies editing methods, because programs are manipulated at a higher level, and operations are implied naturally by pointing and positioning. The system can exhibit more intelligence than conventional systems because as the representation is closer to the machine representation, it is easier to use such things as context to improve interaction (Edel, 1986).

Pict is an interactive graphical programming environment (Glinert, 1984). Pict users never have to touch the keyboard because the programs they compose contain no letters, digits or punctuation marks. Users communicate with Pict through the use of icons. Pict responds by altering its display or providing a help message. It also employs auditory cues to confirm acceptance or error. User programs look like flow charts. Pict developers do not advocate the use of PBD. They believe that both the interaction techniques and graphical nature of PBD systems are the reasons for their success. Pict is designed to be a procedural, algorithmic programming environment in which computer graphics play a central role.

With appropriate design, the visual elements of a visual programming language may be mapped more directly to a programmer's existing knowledge, easing both interpretation and planning (Norman, 1988). If successful, an end-user programming language will remove the requirement that computer experts trained in procedural or object-oriented languages create educational software, and instead enable people who understood how students learn to create it (Finzer & Gould, 1984). Analysis also supported the idea of utilizing an engaging graphical system with easy-to-comprehend interaction techniques as a necessary key in bolstering user interest. A graphical environment would also make appropriate use of other sensory aids, such as auditory cues, as reinforcement (Blattner, Sumikawa & Greenberg, 1989).

2.1.1 Programming by Demonstration

In order to identify a mode of visual programming which might prove particularly effective for the novice programmer population, a second metaphor of visual systems,

Programming by Demonstration (PBD), was also explored. PBD is based on the premise that having programmers show what they want to accomplish should be sufficient to accomplish the task. Programming tasks should be as simple as demonstrating to computers what their goals are. Computers can then add any required details or code to carry out their assigned tasks (i.e. code rendering). This scenario has been labeled "Watch what I do" (Cypher et al., 1993).

The system creates generalized programs from users' recorded actions. This strategy extends the simple approach of recording by working with high level events and generalizing these actions to create programs. For example, a demonstration applied to one simulation element (say, a rock) might be generalized to apply to all other rocks in the system. PBD removes a level of abstraction from programming by allowing users to simply demonstrate their current mental model. Instead of users trying to think more like computers, computers obtain concrete specifications directly from users and translate those demonstrated tasks into appropriate computer models. Activities supported by PBD systems facilitate end-user programming of applications.

Pygmalion is an early example of a PBD system (Smith, 1993). This system introduced the concepts of icons and mixed-initiative computing, and is designed to stimulate people's creative thinking. Mixed initiative or mixed mode computing utilizes both text and direct manipulation styles of programming in a complimentary manner. Users may perform a task by direct manipulation, but may add a level of specialization or add values to variables as text. The system tries to do this as follows:

1. Emphasizing visual man-machine communication through the use of a graphics display.
2. Providing interactive feedback.
3. Enabling programming by demonstration.

Pygmalion uses a blackboard metaphor (i.e., all parts of the program are visible like a physical blackboard). The user interaction technique is simply choosing operations/opcodes from a menu, and customizing them once they are placed on the

blackboard. This illustrates the technique of using instances or concrete examples as surrogates for more abstract concepts (i.e., the class to which they belong); the instance-based interaction style was a key contribution of early systems such as Pygmalion.

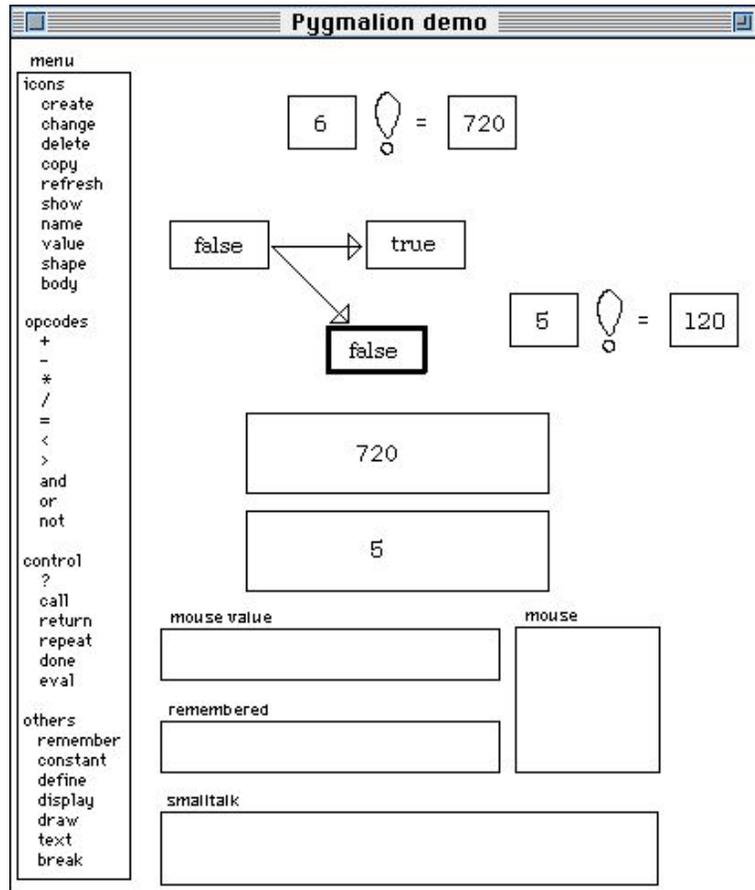


FIGURE 2.3 PYGMALION SCREENSHOT: 6 FACTORIAL DEMO
(*www.pygmalion.ws, 2004*)

Tinker is another PBD system that relies on interaction with concrete examples:

Tinker permits a beginning programmer to write Lisp programs by providing concrete examples of input data, and typing Lisp expressions or providing mouse input that directs the system how to handle each example. It improves Pygmalion's approach where users are forced to anticipate future arguments. The user explicitly indicates which objects serve as examples, and may present

multiple examples that serve to illustrate conditional procedures. The machine records the steps of the computation, and formulates a program to handle the general case. (Lieberman, 1993, p51)

Thus, Tinker adds to the general notion of PBD by expanding the degree of generalization provided and the interaction techniques used to indicate or specify what the user wants.

Macro recorders are used in some systems to allow users to program by example (i.e. recording their actions and replaying them). Rehearsal World (Gould and Finzer, 1984) and KidSim (Cypher & Smith, 1995) both utilize this mechanism. Rehearsal World was an early system built in Smalltalk that was created to provide more control for curriculum designers. Everything is made visible to both remove abstraction and allow designers to think concretely. It extends traditional programming by adding a macro recorder that allows users to create programs without programming. This frees novice programmers from thinking about programming language details.

Rehearsal World also provides a basic troupe of performers that respond to many cues. Novice designers can both use these examples and extend the system by creating new performers. This functionality makes the process of creating a simple production quick, easy, and enjoyable (Gould & Finzer, 1984). Because putting a blank canvas in front of an artist or a blank pad in front of a writer is often an impediment to creativity, this approach of supplying templates is a definite plus. Rehearsal World also outshines Tinker in both the richness of its library of troupes and its informative help system (Gould & Finzer, 1984). Learning by example is a very effective learning strategy, which was another important contribution to this study's approach to building simulations, namely the notion of providing pre-defined actors and specialized training materials. Also the creators of Rehearsal world designed their environment to return control of interactive computer graphics to the curriculum designers:

Design and implementation constitute two phases of a feedback loop. In most design situations, in which programming is a separate and specialized skill, the designer must somehow convey embryonic ideas to a programmer ... Then the programmer goes away to write a program so that something shows on the screen to which the designer can respond. This process introduces interruption, distortion, and delay of creative design. (Finzer & Gould, 1993, p. 81).

Rehearsal World was a prototype system at Xerox PARC, implemented in black and white and not widely available for use, that was utilized by very few designers. The system was implemented in Smalltalk76 and the interaction techniques and programming style, shown in Figure 2.4, is reflective of the underlying language.

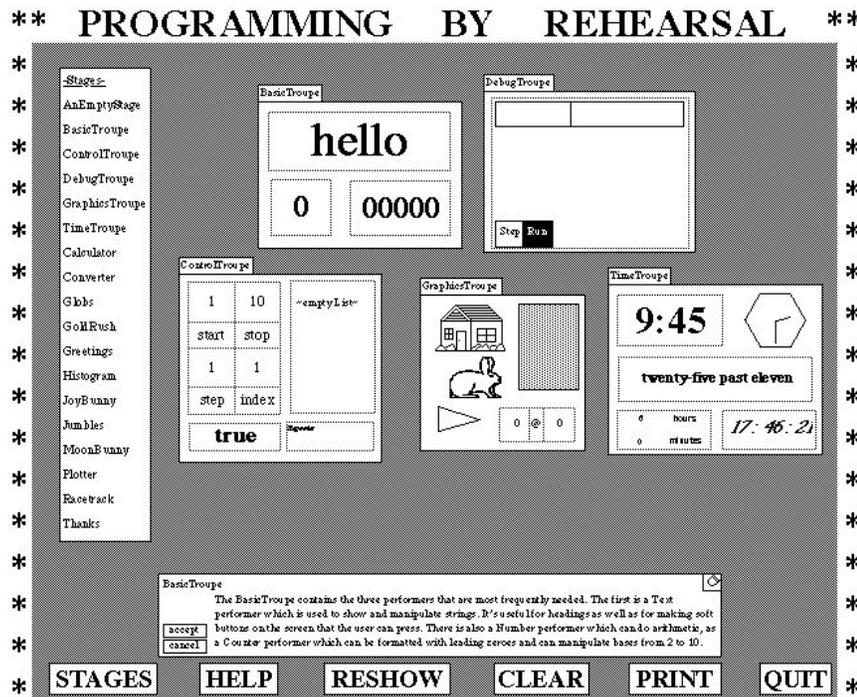


FIGURE 2.4 REHEARSAL WORLD

“The entire Rehearsal world theater, showing the Stages menu at the left, all the available Troupes, and a descriptive help message about the Basic Troupe.” Cypher et al., 1993, p78.

The user interface of Rehearsal World is a window where the user can add predefined troupe windows to the interface. For example, the teacher may select the Graphics Troupe, which contains predefined graphics, and then choose a template

primitive performer to add to a stage. The teacher can utilize the “Basic Troupe” then add questions and include a predefined text area where students can enter answers. Teachers assemble performers on the stage and give them cues to create a performance. All this interface interaction is carried out with direct manipulation, except for the creation of cues. In order to create cues, or rules of behavior, for an object the teacher must specify the Smalltalk code, which they accomplish by initiating a system macro recorder or “watch me” to create the appropriate code by demonstration, and the system then renders as Smalltalk code. “This was perhaps the first example of using programming by demonstration to create actual program code without requiring the user to know the syntax of the programming language” (Potter, 1999). The facility that the user creates code without having to learn a programming language is attractive, but this limits the graphics interaction to simple movements that can be demonstrated.

With Rehearsal World, there is no concept of class. When a teacher copies a performer it becomes a new prototype performer that they can modify and assign a new name. “A major weakness of this method is that modifications made to the first production will not be automatically reflected in the modified one” (Finzer & Gould, 1993, p 92). In Rehearsal World there is also no difference between a class and an instance (i.e. no abstractions or classes). “What is lost is the ability to have changes made to the original reflected automatically in the copies” (Finzer & Gould, 1993, p 93).

Designers create new graphics in this environment via a pixel-based drawing turtle, similar to the LOGO turtle. The teacher designers involved in an initial study spent half a day learning how to draw the background for their game, taking four days to design and implement a game. Since there were few end-users of the system, it was believed that if teacher designers had to manage multiple processes they would soon encounter “conceptual and mechanical difficulties” (Finzer & Gould, 1993, p 99). Finzer and Gould suggested the investigation of a constraint based version of this system in future work. Although the users can create new graphics with this system, this should be an intuitive process and take up much less teacher time. A constraint-based version of a graphical system will be useful in creating intricate and realistic simulations.

Alan Kay (1993) has argued that there are powerful, yet difficult, concepts in programming (e.g., abstraction), and that it is important that EUP does not give up on teaching these concepts in favor of making things easy. KidSim (Cypher & Smith, 1995) works to remove the unimportant difficulties, such as syntax, that impede students so they can work with important concepts like abstraction. KidSim, also known as Cocoa and commercialized as Stagecast Creator, incorporates the use of a macro recorder to allow novice users to program by illustrating what they want to happen in an "example" demonstration.

KidSim's basic control structure is limited to a fixed sequence for testing rules. Rules are tested in production style by starting rule testing at the top of the knowledge base with the first rule in sequence and continuing to test each rule in order until one is fired. Even though KidSim is based on a production model of execution it embodies some important programming concepts, such as conditional execution, subroutines, iteration, and variables. Although rules are executed in production style, the user can add conditional statements to prevent rules from being evaluated if the condition is not satisfied. KidSim's production rules can invoke function calls to escape the regular order of execution and when the function is carried out it will be executed in production style until the last rule in the function is tested. The execution will then return to the calling function after the function invoking statement. Also, the teacher can add variables that can be iterated and utilized in pre-conditional tests for rule execution.

There are many other notable systems that use PBD. For example, AgentSheets is similar to KidSim, but extends it by adding extra functionality and flexibility. Users can utilize the interaction technique of direct manipulation (i.e., dragging and dropping of interface objects) to create agent behavior using options built into the Condition and Action palettes to select program operations. This simplifies users' creation of programs by eliminating the need to understand programming language textual syntax. While using AgentSheets, users are able to add new rules to the behavior palettes, which act as an extension of the environments. Just as in KidSim, Stagecast allows users to create their

own visual characters with a bit map editor. Stagecast also allows the importing of appropriate types of bit map images (e.g., bmp). Both AgentSheets and Stagecast offer users the option of importing any bitmap image or jpeg as a character's visual image, as well as the ability to grab (select) any color from the computer desktop as a resource when editing a character or background image.

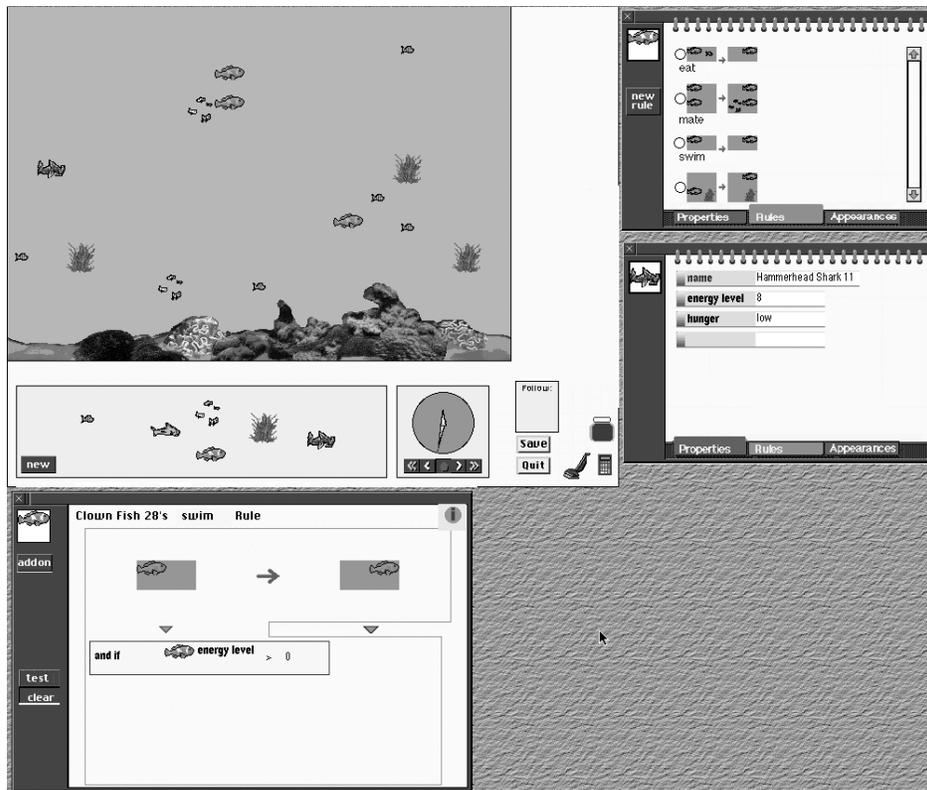


FIGURE 2.5 KIDSIM INTERFACE AND RULE EDITING WINDOWS
 (<http://www.acm.org/sigchi/chi95/Electronic/documnts/demos/ac2fg1.gif>)

To provide more realistic simulations, AgentSheets can simulate non-determinism in actions. Instead of an action being executed at every cycle, AgentSheets can specify that a rule is only executed a certain percentage of the time (i.e. a percent chance rule). This functionality gives object behaviors a more natural appearance that is less deterministic or predictable. With the AgentSheets package, the user is also able to import high-quality visuals as backgrounds. AgentSheets also incorporates the concept of flow (e.g., flow of water, traffic, electricity, etc.). Users can test and run simulations with ease, just as in KidSim. One of AgentSheets' most powerful features is its support of

reusable agents. Individual rules, agents, or entire simulations can be downloaded from a web-based repository called the Behavior Exchange (Repenning, 1995). AgentSheets advocates claim, as a result, that even users with no prior programming knowledge can run a simulation, build a simulation from scratch, or reuse simulations.

Mondrian (Lieberman, 1993) offers a novel extension to the macro recorder paradigm by using synthesized speech to explain its inferences while the user is demonstrating a program. Garnet (Myers, 1990) (built from Peridot (Myers, 1987), now Amulet) provides a complete User Interface Development environment that allows the effective use of inferences, of PBD, and of special purpose editors and dialogs.

This review of PBD systems reinforced the advantages of providing users with experiences that promote learning by example. PDB also helps novice users begin to create interactions very quickly, with much less cognitive overhead than learning an entire textual programming language. Although this study focused on examples designed for a learning and reuse environment (e.g. example simulations to provide salient examples for reuse), any continuation of the work should also consider example-based resources more generally.

2.1.2 Simulations/Construction Kits

“Computer based simulation or modeling is the creation of a computer program, which attempts to simulate an abstract model of a particular system and is very useful in modeling many natural systems in physics, chemistry and biology, human systems in economics and social science and in the process of engineering new technology. Traditionally the formal modeling of systems has been via a mathematical model, which attempts to find analytical solutions to problems which enable the prediction of the behavior of the system from a set of parameters and initial conditions. Computer simulations build on, and are a useful adjunct to purely mathematical models in science, technology and entertainment” (www.wordiq.com/definition/Computer_model). The systems that we study are not purely mathematical models of phenomena, but in many cases metaphorical, virtual, or visual models of natural phenomena (i.e. an exemplar of

an earth science and/or and physical science system). This type of simulation might be termed as light weight simulations that simply help the user to study a dynamic system. When we use the term simulation we speak of an educational program that is used to construct simulations that model an educational episode or content example lesson, for educational study.

A number of systems have been developed specifically with simulation construction in mind, using a “kit” metaphor in which users instantiate and interconnect predefined elements. Some of the key systems in this class are SimCity (www.maxis.com), Pinball Construction Set (Electronic Arts), Star Wars Droid Works (Lucas Learning), ActivChemistry (salamanderinteractive.com), and Geometer's Sketchpad (Jackiw, 1992). SimCity, Pinball Construction Set, and Star Wars Droid Works are games that allow users to create interesting simulations. All of these systems are End user Programming tools that students and teachers can program with no prior knowledge of a programming language, but use of these tools is limited to preprogrammed behavior.

These programs are very good for educational lessons in the specified domains. Indeed, a completed AgentSheets or Stagecast "program" can be viewed as a construction kit. Students are challenged to use the simulation in creative ways in order to succeed. The approach can be likened to an educational video game in that there is a fixed set of resources and types of interactions that can be employed to learn about the environment. ActivChemistry is an educational construction kit that provides a chemistry toolkit for building simple chemistry experiments that can also be used to give students drill and practice on chemistry concepts and experimental methods. Geometer's Sketchpad (a modern extension of Ivan Sutherland's historic Sketchpad system) supports the construction and manipulation of geometrical forms in an interactive geometry environment.

supports visual interaction in programming by using several types of icons, selecting icons from a menu, and connecting them. ThingLab employs a constraint model rather than an imperative model, and while constraints may be represented graphically (Borning, 1981), methods for resolving the constraints are displayed textually.

A study of two commercial visual object oriented programming environments was performed to determine whether visual programming could be successful for realistic programmer applications. The systems that were studied were National Instruments' LabVIEW and Hewlett Packard's Visual Engineering Environment (VEE). The study investigated the creation of programs by manipulating and connecting icons, instead of traditional text-based programming, finding that merging visual and object-oriented techniques offers significant gains in productivity and system reliability. Visual syntax techniques also led to better requirements, better communications between developer and customer, and allowed code to be created interactively with the customer by using visual object oriented programming tools (Baroth & Hartsough, 1994).

In a study of PROGRAPH, evaluators explored the use of developing methods pictorially as opposed to traditional language design. They first explained that text is clearly very important in a programming environment, and is superior when expressing concepts that are inherently textual such as algebraic formulas and to distinguish objects of the same type:

For example, in PROGRAPH, textual names are used to identify methods of the same class. Comments obviously require text, since the information they convey is informal and explanatory. In general, pictures provide a better representation of most complex structures, such as algorithms, since they correspond more closely with the mental model of these structures. Expressing such things as text introduces an unnecessary layer of abstraction which must be removed by parsing the text to mentally construct a picture. (Cox, Giles, & Pietrzykowski, 1994, pp. 63-64)

This study of visual object oriented programming serves the following two purposes: 1. to identify better methods of system design and development, and 2. to utilize a means of environment presentation and simulation creation tools that is expected to give our participants the ability to create programs with less cognitive baggage and fewer layers of abstraction for understanding and creation.

2.3 Software Reuse

“The best way to attack the essence of building software is not to build it at all. Package software is only one of the ways of doing this. Program Reuse is another” (Brooks, 1995, p. 222). Program or software reuse can be carried out in many ways and some methods of formal reuse and informal reuse are discussed in this section. There is an anecdote about reuse: “Would you be the first passenger on an airliner whose parts have just come out of the R&D shop? Or would you prefer to board knowing the aircraft was designed and constructed with parts that have successfully kept planes airborne for years? Practicing engineers... select from components whose characteristics have been tested and proven for safety and efficiency. Reusability is a fundamental principle of engineering” (Williamson, 1997). Using strategies from formal software reuse, most successful large scale software systems are created from previously tried and tested code modules, which is referred to as software reuse. The practice of formal software reuse reduces the time to create and test applications and improves overall software quality. Another advantage is that reusing code is an important way to reduce software costs and thus make programmers and designers more efficient (Johnson, 1991). Professional programmers make use of standard libraries and repositories as the foundation of their work, and then refine or refactor the code based on the specific program requirements. The promise of easy reuse of classes, with easy customization via inheritance, is one of the strongest attractions of object-oriented techniques (Brooks, 1995), and object oriented programming (OOP) is well suited to formal software reuse support. It will thus provide a very supportive technique for the building of our environment.

Professional programmers and program designers rely on these formal methods to improve their work practice, but they also incorporate informal reuse methods. The practice of informal software reuse describes the process whereby the programmer simply looks around for useful examples that can be copied and modified or used as models. In this study, software and component reuse techniques will be used to assist our teachers in the development of their educational simulations, as it is often more efficient to reuse code than create it from scratch and software reuse is an advantage of the OOP paradigm. In a study designed to develop a qualitative characterization of expert informal reuse strategies that could be used to identify requirements for teaching and supporting reuse programming, several strategies for expert reuse were found that may also be employed to train less experienced programmers. They include assessing similarity, studying context (e.g., reasoning by analogy from familiar syntactic construction), and by testing (Rosson & Carroll, 1996b).

In many cases, it is not necessary to build software completely from scratch. “Most experienced programmers have private libraries, which allow them to develop software with about 30% reused code by volume. Reusability at the corporate level aims for 75% reused code by volume, and requires special library and administrative support” (Jones, 1995, p. 222). However, even computer science departments have a need to create software. Programmers build from scratch for many reasons, including wanting something for a specialized application, not being aware that what is wanted has already been created, because they can, and because they generally are only paid to write code and not to reuse it. Programmers need to foster good strategies for software reuse and understanding software frameworks to end the cycle of reinventing the wheel with every software project. Software reuse and the OOP paradigm are attempts to educate users, helping them recognize common standards, frameworks, or patterns for software components. This can save time by avoiding redundant work (Roschelle et al., 1999).

Looking at strategies used by professional programmers, both formal and informal, is helpful in developing a strategy of reuse to support novice programmers in learning reuse techniques and using these techniques to improve their work practices. For

example, a common novice reuse technique is an informal strategy of reuse caused by the production paradox, “the copy-paste dilemma”. When novice users are presented with the need to create an artifact for a simulation, the user as a first technique copies from an existing artifact, pastes it to reuse it, and then specializes it as needed. Supporting this type of reuse style would compliment current user practice.

Because teachers have so little time to learn and use new technology, it is important to provide reusable components for them to apply. A formal reuse strategy that would be helpful in a novice reuse environment would consist of an environment, training materials that encourage them to create, reuse, and modify existing components to reduce creation time, and support for their day-to-day activities. With the growth of the Internet and the World Wide Web, many specialized online communities have begun to emerge and the professional educators' community is no exception. As the Internet has become accessible to more and more people, a number of communities have developed to provide support for teachers and help to improve teacher practice. These Internet communities are gathering places designed to support creativity and encourage participation. These communities include Educational Object Economy (EOE) (www.eoe.org), Educational Software Components of Tomorrow (ESCOT) (www.escot.org), and TAPPED IN at the Stanford Research Institute (www.tappedin.org). Among the largest educational communities are EOE and TAPPED IN, which invite teachers and developers to collaborate to learn and share their experiences with technology.

The screenshot shows the ESCOT website interface. At the top, there is a navigation bar with the ESCOT logo and links for 'overview', 'research', 'about us', and 'sitemap'. Below this, a sidebar contains a menu with icons and labels: 'interactive problems', 'authoring teams', 'component tools', and 'math standards'. The main content area is titled 'Educational Software Components of Tomorrow'. On the left, a 'Our Mission' section describes the project's goal of integrating technology in middle school mathematics. On the right, three sections describe the roles of 'Classroom Users', 'Developers', and 'Teacher-authors'. In the center, a large graphic features several concentric red arcs over a teal background, with a red triangle at the bottom labeled 'Lift Off' and a red cross in the upper right. Below the graphic, the text 'Featured ESCOT Problem of the Week:' is displayed.

FIGURE 2.7 ESCOT
(www.escot.org, 2004)

Most professions have organizations that promote professional development, but most teachers do not have access to such resources because they work in isolation in their classrooms and have few other individuals in their content area within their schools. A local effort, the LiNC (Learning in Networked Communities) project, has emphasized a complementary view—the need for local community support for teachers (Carroll, 2000; Koenemann et al., 1998). In most professions, there is peer support for professional development activities with face-to-face meetings where many aspects of work and life are shared, including the sharing of knowledge and resources. In teaching, the opposite is usually true. Many schools may only have one or two teachers in an entire school that teach a given grade and content area.

The new online educational communities provide teachers with many opportunities for discourse and educational exchange. As a result, they have been used as organizers of professional development activities. They provide arenas for sharing ideas and building relationships that would normally take place in meetings conducted in person (Schlager, Fusco & Schank, 1998).

For example, ESCOT's goal is to become a premiere site for educators and developers to store and create diverse educational software components that will work together through a common architecture. Every product created in this collaboration is integrated into the organization's database in the hope that everyone will use a common interface to make the software more universally compatible.

The current focus of ESCOT is mathematics education. The coalition goal is "not the creation of a single software product, but an understanding of how 'integration teams' comprised of developers, authors, teachers, web facilitators, and others could compose lessons by combining graphs, tables, simulations, algebra systems, notebooks and other tools available from a shared library of reusable components. Our integration teams draw upon Java versions of such powerful tools SimCalc MathWorlds, Geometer's Sketchpad® and AgentSheets to begin addressing the needs of five new middle school mathematics curricula for empowering web-based learning technology." (www.escot.org/External/background.html). This is an exciting vision. Participating in a community such as this will be of great benefit to those who wish to either create or use reusable educational software.

The ultimate goal of this research is to provide aids to professional development for local teacher communities, including providing an online repository for teachers to store and reuse simulations and simulation components, collaborating with a group to create components for the online repository, and studying the impact of this repository on creating a virtual community of local peers. The results will be integrated into a simulation community such as Tapped In or ESCOT. The shorter-term focus, however, is

to enable a process of informal reuse through well-designed example components and simulations.

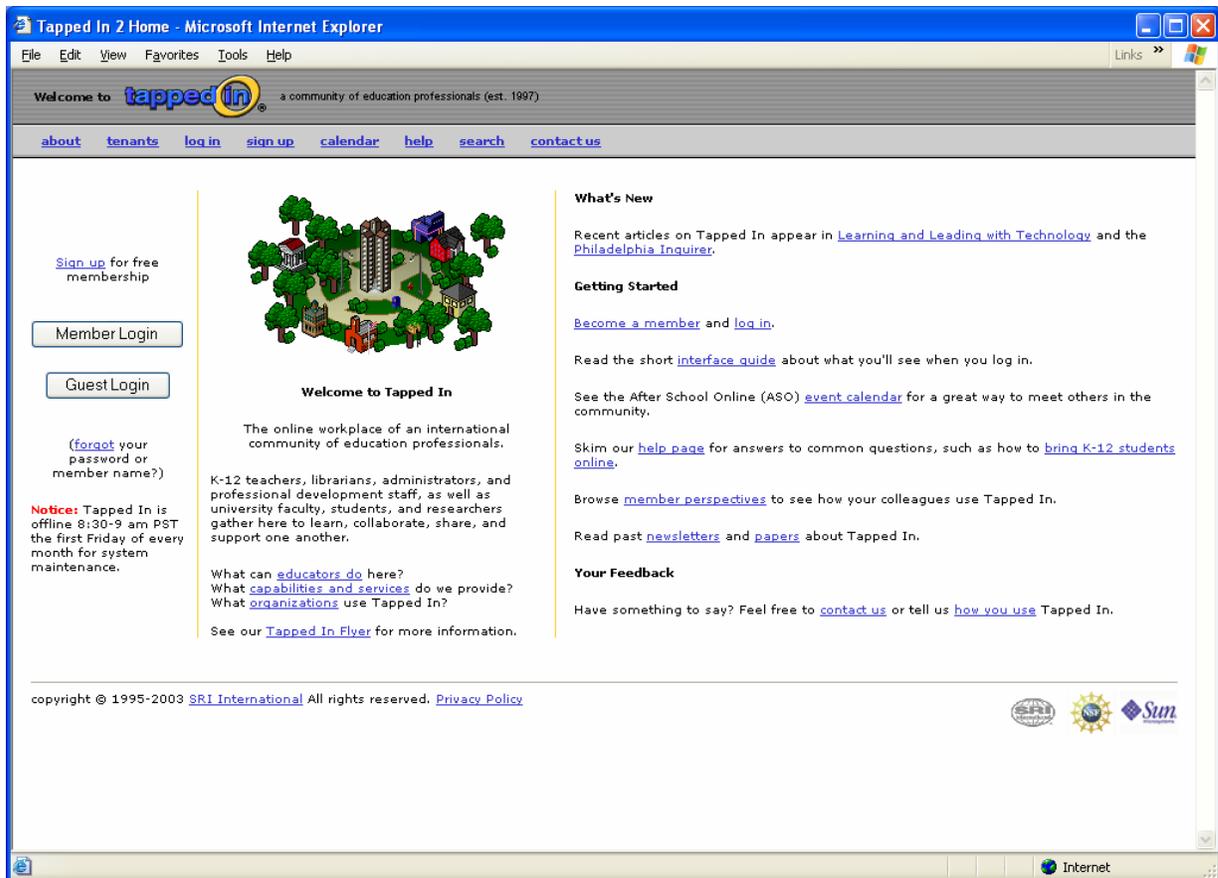


FIGURE 2.8 TAPPED IN AT STANFORD RESEARCH INSTITUTE
(*www.tappedin.org, 2004*)

2.4 The Use of Simulation Tools by Teachers

Assisting teachers to learn to use simulations could be a great boost to the introduction of new software applications as curricular aids, improving education by giving students exercises that are more compelling than traditional textbook and chalkboard examples. There are more computers in the classroom than ever before, but there is very little direct evidence of teachers learning to use simulations and very few studies that have investigated teachers' ability or willingness to learn and use simulation

tools as part of their practice. With respect to teachers learning technology, there are only a few studies of novice programmer teachers in support of teacher learning and adoption of educational software, and most studies were exploratory in nature. The studies identified are discussed below and include: a study of teacher designers designing simulations (Finzer & Gould, 1984), a study of teacher adoption of new technology (the MVHS project report, 1997), a study of experienced teacher designers who were familiar with computer graphics (Goos & Geiger, 1999), and a summary of teacher experience when working to adopt new technology (Stoner, 1996).

The goal of the first study was to design an environment tailored specifically for educators. Teachers used an environment to develop educational software, “Programming by Rehearsal”, and the study explored whether computers could be used to improve analytical and cognitive abilities, and also enhance education through non-analytical, intuitive thought (Finzer & Gould, 1984). The system was intended to allow non-programmer curricula designers to implement their own ideas and use computers to support graphic and intuitive thinking in the same way as books have traditionally been used to support linear and verbal thinking. Giving educators more direct control was expected to create high quality educational software or, at a minimum, elicit better requirements.

Finzer and Gould (1984) found that graphical programming environments could facilitate the creation of interactive graphic programs, and that the visual paradigm held out some hope of supplanting the current process of textual code. Everything in Rehearsal World is visual, ensuring the designer thinks in concrete terms. The study concluded that beginning designers found the concreteness of their representation helpful, while more advanced designers benefited from more general and abstract terms, and at some point (e.g. access to large amounts of data) the concrete nature might prove a barrier to program complexity. Also, this system does not allow the designer to add new types of actions for objects and does not support constraint-based problems. This study showed that teachers will use software to create activities for their classrooms, but the usability of

the environment must be increased and the amount of time needed to realize an activity or simulation decreased in order to be attractive for teachers.

A study at Maryland Virtual High School dealt with access to technology, teacher adoption of new technology and experimentation with new curricula. The proposal was that “peer and mentor support, additional technical advice, technical resources and a source of problem-solving activities related to science and mathematics as well as coaching by master teachers can help to equalize the learning for all levels of students” (MVHS project report, 1997, p2).

This three year study was very successful, but most school systems do not have the resources to provide all of the additional training and support required. Even in MVHS, only about 20% of the classrooms had three or more computers in the room, with the remainder making do with one or two and limited access to the school’s computer labs.

This study had to overcome many hurdles, including limited technical expertise and the need for more support and computer infrastructure to support the new curricular activities. Once teachers begin to utilize new activities and learn new technologies, the hope of educational software creators is that teachers will continue to use the new technology and find it useful in their classroom practice. The study found that it takes at least a year for schools to become familiar with new infrastructure. To compound the problem, classroom materials had to be developed by teachers learning the new computational science paradigm and teachers felt stimulated, yet frustrated; excited, but exhausted. Teachers have little free time and an environment with excessive start up time will waste valuable time to attain goals that teachers initially perceive to be of little value and merely an extra burden on their schedules.

Over the three years of the MVHS project, with fifteen schools involved, only a dozen teachers were able to create computational science units. This is evidence that even though there was technology support and software available, educational software needs

to have improved usability and learnability. Our aim in this study is to reduce this frustration and the exhaustion experienced by teachers by providing an educational software creation toolkit and an accompanying minimalist instructional manual that provides self paced instruction, contains a minimum of textual instruction, and gets teachers started much more quickly with support of error recovery. The time and effort that teachers had to expend in the MVHS project adversely affected the success of the venture. Systems of instruction and support for teachers in the use of educational software need a much lower learning curve and support quick start up.

Goos and Geiger (1999) dealt with the need for professional development to enhance teachers' competence and confidence in using technology. They focused on observing how students used a range of technologies, and discussed the implications for the teacher's role in a technology rich classroom (i.e. equipped with computers and scientific calculators). In one lesson, the task was to find a solution for a complex mathematical equation. The students' first attempt to solve this equation was by using their graphing calculators, which found only a partial solution. The students confirmed to the researchers that they would have been satisfied with the initial solution and would not have used the spreadsheet software to find additional solutions if not directed to do so by their instructor.

During another class exercise, students had to choose between two calculators. The students indicated that the most important factors in their choice were their familiarity with the calculator, its user friendliness, screen size and resolution. For this exercise, all the students were instructed to create the same program. The teachers downloaded student solutions to his viewscreen calculator and shared their programs with the whole class using an overhead projector. The formula used was the same for all equations, but there was a wide variation in the other elements of the students' programs, which generated a lively discussion. This provided an opportunity for students to reflect on their learning and learn from the solutions of others. One student commented, "Technology enables us to learn for ourselves and the teacher is still there to assist, but we do the initial thinking and working. ... [This is] beneficial for us as we can learn from

our mistakes as we go along. This actually makes the role of the teacher more important, as they can assist us and advise us about how to do things, rather than the usual dictator of the past". Work like this re-examines the role of the teacher and suggests new types of interactions between students and teachers. Also, it calls for students to have more ownership of their educational experiences and be reflective learners. Simulation creation and exploration can be very supported in reflective and inquiry-based education.

The Learning Technology Dissemination Initiative (LTDI) (Stoner, 1996) is expected to support and encourage the integration of technology into higher education teaching. It was created to help staff identify areas in the curriculum where computer technology could be added to improve courses. The plan includes extensive support for teachers in the form of workshops, use of the World Wide Web, email groups, video conferencing, and a resource collection. This project has yet to be realized and its main goal is to inspire teachers, encouraging them to evaluate their user of technology in the classroom and to review their personal usage of technology. Stoner also suggests extensive strategies for technology integration and the eventual evaluation of the initiative.

This review confirmed that there have been very few studies of teachers learning technology. However, when technology is used in the classroom it can be very beneficial for student learning. These prior works identified strategies that aid teachers in choosing new technologies for their classrooms and suggest methods to infuse technology within their classrooms. Also, the foundational work of the LTDI, although not in the U.S., provides a comprehensive blueprint for technology adoption and will be very useful when drawing up plans to provide teachers with future support when introducing new educational technology.

2.5 Human Computer Interface Design

Design is hard. Fred Brooks states, "There is no single development, in either technology or management technique, which by itself promises even an order-of

magnitude improvement within a decade in productivity, in reliability, in simplicity" (1995, p.179). In other words, there is no "silver bullet" or "one size fits all" recipe to make interface design and implementation more tractable. Brooks goes on to say, "Not only are there no silver bullets in view, the very nature of software makes it unlikely that there will be any" (1995, p. 181). There are many difficulties associated with designing complex systems, and user interface design increases the complexity by adding the following problems (Meyers, 1998):

- Designers have difficulty in learning users' tasks
- Tasks and domains are complex
- There are many different aspects to the design which must all be balanced, such as standards, graphic design, technical writing, internationalization, performance, multiple levels of detail, social factors, legal issues, and implementation time
- Existing theories and guidelines are not sufficient
- Iterative design is difficult

Meyers also argues that user interfaces are especially hard to implement because:

- They are hard to design, requiring iterative implementation
- They are reactive and must be programmed from the "inside-out"
- They inherently require multiprocessing
- There are real-time requirements for handling input events
- The software must be especially robust while supporting aborting and undoing of all actions
- It is difficult to test user interface software
- The tools to help with user interfaces are extremely complex
- Programmers report an added difficulty of modularization of user interface software

One approach to simplifying the design of interactive systems is to use prototyping tools that support iterative design. For example, interface prototyping tools such as Apple's HyperCard and HyperStudio, Microsoft's Visual Basic, and

Macromedia's Director, might allow rapid development and refinement of simulation environment prototypes. However, research has shown that production-quality interface prototyping tools such as these can over-constrain design. Traditional user interface tools force designers to bridge the gap between their plans for a design and the detailed specifications they must create using a tool (Landay, 1996). Further, most of today's toolkits and interactive tools are still quite hard to use and lack flexibility (Meyers, 1998). Because the research goals of this study were quite open-ended and exploratory, the use of rapid prototyping tools was rejected as being too restrictive.

Another strategy for improving the design of interactive systems is to apply guidelines for user interface design. For example, a review of general issues in user interface design is invaluable when creating the new visual programming environment. In general, Shneiderman's "Eight Golden Rules of Interface Design" (1998) provided the development guidelines used, namely:

1. Strive for consistency
2. Enable frequent users to use shortcuts
3. Offer informative feedback
4. Design dialogs to yield closure
5. Offer error prevention and simple error handling
6. Permit easy reversal of actions
7. Support internal locus of control (want to feel in control)
8. Reduce short-term memory load

Dykstra et al. (2001) state that all designers, whether formally trained or not, struggle with the fact that a good design that makes a positive and significant difference in everyday life, is almost a miracle when actually developed. Cross (2001), however, suggests that design ability can be fostered. All designers should be encouraged to share their perspectives to both improve design and bridge the gap between research and development.

2.6 Minimalist Instruction

Given the general demands on teachers' time, it was necessary to find an instructional method that would aid in training teachers for this new educational software environment in a timely manner for this study. Most of the systems approaches to training were too extensive for the specialized needs of teachers and would not be suitable for this study. Instead, the minimalist model of instruction (Carroll, 1990) was applied to create a specialized introduction to the environment with our teachers' needs, time constraints, and motivational obstacles in mind.

The term minimalism was first coined by John Carroll and his associates in the 1980s, and was more formally introduced in *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*. Carroll stated:

The goal of minimalism is to teach people what they need to learn in order to do what they wish to do. We attempt to design instruction to suit the learning strategies people spontaneously adopt, and are empirically driven by what people want to do, how they want to do it, and the problems they are observed to experience. When contrasted directly with instruction developed in the systems approach, our instructional designs have produced faster and more successful learning. Drill and practice has failed, users ignored steps and directions of the training that seemed irrelevant to their task-oriented concerns (1990, p. 5).

The main aspects of a minimalist approach include:

- Learners are allowed to start immediately on meaningfully realistic tasks
- The amount of reading and other passive activity in training is reduced
- Errors and error recovery are less traumatic and more pedagogically productive
- There is more interest in action and in working on real tasks, than in reading. Thus, an instructional designer can better meet learners' needs by

presenting material once, as briefly as possible, and as a result make the training as unobtrusive as possible.

Carroll also outlines a set of minimalist instruction principles regarding information design:

- Use a simple example based on prior knowledge
- Minimize the text to be read by the learner, which reduces reading time and leads to incomplete directions and explanations that in turn require the user to reason and improvise to accomplish and understand the programming task
- Include open-ended questions in each block to stimulate self directed exploration
- Describe the consequences of an action to support error recognition and help user recover
- Use a scaffolded example that builds upon itself so that learners can use their growing understanding as a knowledge base while learning continues

An example of minimalism applied to programming instruction is reported by Rosson, Carroll & Bellamy in their study of Smalltalk learning (1990), who developed a set of example-based learning scenarios aimed at supporting real work, getting started fast, reasoning, and improvising. Their materials sought to address problems such as the lack of familiarity with Smalltalk's extensive class library, navigation within Smalltalk's library, understanding computation as a series of messages sent among objects, and the use of a powerful but complex interactive programming environment. Their supposition was that if instruction is very directive and complete, learners may simply follow instructions blindly without noting the consequences their actions have for system states. Their plan was to give users more open-ended tasks to see whether they could creatively complete the task, actively contemplate the consequences of their actions, and actually learn something.

A second example of minimalist instruction can be seen in the work of Wissman (2002). This study, part of a project on the use of simulations as an avenue for community outreach programs, was an empirical attempt to ascertain the best way to teach computer-related information to older adults. Tests were performed with two different types of training materials, one using a systems approach and the other based on the minimalist approach. The results included recommendations for designing better minimalist instruction for older adults. In keeping with minimalist instructional training guidelines, Wissman created training material that was written with minimal wording but was still able to get the point across. A key consideration was how to motivate learning by older adults in a visual programming environment (i.e., what would be seen as a "real world task" in this environment). Wissman concluded that learning could be promoted as a vehicle for interacting with young people on community-related simulations, a task that the older adults readily accepted as a meaningful goal.

Wissman found that age and experience were the best predictors of performance, with decreasing age and increasing experience both being associated with increased performance. Supporting the minimalist instruction technique, participants in the minimalist group spent significantly less time on the tutorial than those in the control group. Interestingly, females gave identical subjective ratings regardless of the training condition, but males tended to report themselves more satisfied in the control condition than in the minimalist condition. However, the difference was not significant.

Wissman's recommendations were that older adults may require more scaffolding and "training wheels" (Carroll, 1990) for reassurance than normally required for other target groups, because older adults generally have higher levels of computer anxiety than younger people. An ideal training design may be a minimalist approach that provides more frequent feedback and a greater amount of structure, or guidance, than ordinarily found in a minimalist tutorial. This type of design would, hopefully, retain and/or enhance the performance benefits of minimalism while capturing higher subjective ratings by older adults.

In general, studies of minimalist tutorials have been promising, particularly for young people who are generally highly motivated to learn new computing systems and relatively confident about their own skills. Although older in-service teachers may be less confident than young people about their computer skills, they are increasingly experienced in the use of a range of technologies for education. Teachers also possess considerable task expertise, for example, understanding how and why a volcano might be used to illustrate principles of earth science. Thus, it seems likely that minimalist instruction would be effective in helping teachers rapidly gain competence in a new software environment.

2.7 Usability Evaluation

Given the exploratory nature of the proposed work, where little was known in advance about teachers' needs and abilities in simulation programming, it was clear that an iterative user-centered design process would be appropriate (Gould & Lewis, 1987). This, in turn, led to an important emphasis on usability evaluation as a means for obtaining information about users' needs and preferences. Chapter 3 will include an in-depth discussion of a variety of methods, both analytic and empirical, that were used in this study.

Scriven (1967) offered a classification of evaluation methods according to their goals, with *formative evaluation* methods used to guide a design process and *summative methods* used to assess the outcome. He also made a distinction between empirical methods that gather data from users, which he referred to as pay-off methods, and analytic methods that rely instead on expert opinion or review, which he called analytic methods. Carroll & Rosson (1995) described how these two distinctions can be used to plan and manage evaluation needs during the lifecycle of a system, giving examples of how formative goals might be addressed by both empirical (e.g., think-aloud studies) and analytic (e.g., cognitive walkthrough) methods. Summative goals may also be addressed by both empirical (e.g., a controlled experiment) and analytic (e.g., a feature count) methods.

The analytic method of claims analysis (e.g. scenario and claims analysis) generally allows the experimenter to utilize less time and money than is required to perform an empirical analysis. In claims analysis:

It is not necessary to observe or interview subjects to evaluate an interactive application. A set of claims can be used for formative goals to increase understanding about the system and guide redesign. Claims can also be used to meet summative goals – the design rationale provides one view of how well a system meets users' needs. (Rosson & Carroll, 2002, p. 232)

Nielsen (1994) suggests that heuristic evaluation by many evaluators can be used to identify as many problems as possible. Another type of evaluation, a pluralistic walk-through (Bias, 1991), has users, developers, and usability engineers analyze a system together to more effectively identify a comprehensive set of problems. A cognitive walk-through (Polson, 1992) is also a good analysis method to use to identify user goals, expectations, and reactions to tasks. These usability inspection methods are useful ways to save time and money during inspection, but there are some tradeoffs. For example, “usability checklists and inspections can produce rapid feedback, but may call attention to problems that are infrequent or atypical in real world use” (Rosson & Carroll, 2002). To refine their inspection efforts, some experts classify problems according to their severity. This severity classification gives the designer guidance in prioritizing the resolution of problems.

Another distinction often made among evaluation methods is between qualitative and quantitative methods. Qualitative methods yield rich descriptions and thus are most likely to be useful in the formative stages of development, when the most important priority is to gather feedback that can guide design. Quantitative methods produce numerical data that can be analyzed formally through statistical techniques for purposes of hypothesis testing. Although quantitative data can be helpful in guiding design (e.g., slow learning times, or low preference ratings from users can suggest changes that should

be made), it is particularly useful for summative evaluation goals, where it is more important to assess how *well* a system serves its users and their needs.

Michael Quinn Patton gave a broad definition of evaluation as “any effort to increase human effectiveness through systematic data-based inquiry” (1990, p.11). He went on to explain: “Qualitative methods permit the evaluator to study selected issues in depth and detail. When you are able to approach fieldwork without being constrained by predetermined categories of analysis this contributes to the depth, openness, and details of qualitative inquiry” (p. 13). Patton’s body of work, including *Qualitative Evaluation and Research Methods* (1990), *Practical Evaluation* (1982), *Creative Evaluation* (1981) and *Utilization-Focused Evaluation* (1978), provided the groundwork and guidelines for maximization of the qualitative aspects of this study.

Patton urged evaluators to use both quantitative methods to obtain subjective user reactions and qualitative methods to explore details or reveal the rationale behind responses to further support or clarify participants’ reactions. Using both methods assists researchers with building a more complete picture of the experiment results. He provided an example of how to combine closed-ended questions with open-ended questions for a more complete interpretation. “The closed ended questionnaire provided data on statistically generalizable patterns, but the standardized questions only tap the surface of what it means for the program to have ‘great perceived impact’” (1990, p. 23). Reducing the risk of bias in the evaluation was also a concern. At times, data can be assumed to be biased, rigged, and even easily dismissed as loaded questions. To avoid this, the addition of qualitative open-ended responses using the participants’ own words integrated face validity and credibility into study results. “Adding a much smaller sample of open-ended interviews also adds depth, detail, and meaning at a very personal level of experience” (Patton 1990, pp. 17-18).

In addition to considering these general issues of concern to usability evaluation, several more specific methods were used to manage the design and evaluation process in this study. The battery of tools utilized in the research effort included: usability

inspection, cognitive walkthrough methods, participatory design, scenario and claims analysis, think-aloud experiments, user surveys, expert review and guideline review. The next section discusses details of these methods and how they were applied in the design and evaluation process.

This research began with informal usability inspections of visual programming environments. Informal inspections, as opposed to formal usability inspections, were utilized because in formal usability inspections the design team must meet with a panel, where a moderator and a room of experts hold a courtroom style meeting to discuss positive and negative aspects of the interface. Although this provides excellent feedback about the interface, it is generally very costly in both time and personnel resources. Thus, for this study the researcher discussed features of the interfaces with a small group of designers, generally consisting of only two or three individuals (Shneiderman, 1998).

Once the informal usability inspection was complete, cognitive walkthrough methods were performed for two to four environments. A cognitive walkthrough consists of experts simulating users walking through the environment and carrying out typical tasks. This technique was used to rate the ease or difficulty of completing a task in each environment to help narrow the field of systems to be studied (Wharton, Rieman, Lewis, & Polson, 1994).

Three methods that can be very helpful in usability design and evaluation are participatory design, scenario and claims analysis, and think-aloud experiments. In participatory design (Olson & Ives, 1981), the designer and an intended user of the system work together to complete a scenario walkthrough, while the designer records information about the session in order to improve the design of the system. The end-user must commit to several sessions with the developers, and the study will record user insights about their current practice (Shneiderman, 1998, Carroll & Rosson, 2002). This technique will be used as a longitudinal study, with the researcher only observing with no interaction with the user.

In scenario and claims analysis, the designer chooses a set of typical tasks that the user would undertake in the environment. With those tasks, the user is given a task goal and the scenario explains the task the user must undertake. Based on a walkthrough of the scenario, the designer will identify claims for each task scenario. The claims are denoted as positive claims (i.e. +, or pro) and as negative claims (i.e. -, or con) and they provide a means for scenario based design and analysis. The developer would generally see the pros as validation of their designs, and the tester or investigator would see cons, or negative claims, as areas that need improvement in the interface design. This will be a very effective method of comparing the designs of the two systems in this study (Carroll & Rosson, 2002).

Also during the empirical studies, the think aloud protocol will be utilized. This is a usability method where as the user is performing the experiment, they provide a running commentary, asking questions about the experiment or system to give the facilitator feedback. Users verbalize their goals, plans, reactions, and concerns as they work through the test tasks (Ericsson & Simon, as cited in Carroll & Rosson, 2002, p. 243).

Some methods that will be helpful in the study's analysis are user surveys, expert review and guideline reviews. User surveys are a widely accepted and inexpensive instrument to use in concert with an empirical test in order to gather insights about the exercise. The user surveys for this study are a combination of user satisfaction and acceptance tests and are used to gauge how the user rated his interaction with the environment. Before the users undertake the experiment, they will complete a user background survey or questionnaire. After completing the experiment, they will complete a user acceptance test. The survey used here was based heavily on the Questionnaire for User Interaction Satisfaction (QUIS), which was developed by Shneiderman and refined by Norman and Chin (Chin & Rosson, 1988). This survey was based on the OAI model and covers interface details, interface objects, and task issues (Shneiderman, 1998). The questionnaire was used in various ways, and adapted a subset of the questions in order to tailor it to the study's environment and specific needs.

Of the many types of expert reviews (such as guideline review, cognitive walkthrough, formal usability inspection, and heuristic evaluation), the study utilized guideline review as the final expert review. In guideline review, the interface is checked for conformance with a list of guidelines (Shneiderman, 1998). This provides an expert comparison of user created artifacts in order to assess which environment produces better reviews based on the guidelines.

The diverse methods for usability evaluation applied during the design and analysis of the study's environment are expected to provide sufficient data to help shape the design, development, and usability evaluation of this environment.

2.8 Summary

This literature review has summarized the work of many researchers who were pioneers in the area of visual programming environments, as well as systems that serve simply as visual front ends for textual languages. This includes environments that use dataflow, actors and agents, systems used as user interface development environments, and systems that support a constraint-oriented approach.

In addition to pursuing specific interests in visual programming, construction environments, novice programming techniques, and teacher communities, OOP, software reuse, human-computer interface design, minimalist instruction, and usability evaluation were also reviewed. These areas contributed general background and motivation to the process of designing and building a robust system that will be both usable and useful for novices.

CHAPTER 3: RESEARCH OVERVIEW

In this chapter, the research problem is introduced, along with the associated questions and hypotheses addressed by this study. The characteristics of the empirical research context that are general to all of the studies described in subsequent chapters will also be described in detail.

3.1 Problem Statement

In order to use simulation software effectively in their own classrooms, teachers must be empowered to create or modify simulations that suit their curricular needs. Unfortunately, teachers have little time or motivation to learn new software systems, much less develop and refine their own educational simulations. A promising approach for providing more useful curricular aids in the form of educational simulations is the use of visual simulation environments developed for end-users (e.g., AgentSheets, Stagecast Creator). However, to date such environments have been designed largely for students and have not been evaluated for use by teachers. The aim of the current work is to study teachers' learning, usage and reuse techniques in visual simulation systems so as to better understand their needs; to build a new environment that addresses those needs; and to evaluate the effectiveness of this environment in comparison with a state-of-the-art alternative.

3.2 Research Approach

This study utilized a combination of analytic and empirical methods to evaluate the use and development of visual simulation languages and tools. This work can be summarized as three phases, illustrated in Figure 3.1. In Phase I, the requirements analysis phase, initial requirements were gathered based on an analysis of existing languages and environments. Once the informal usability inspection of a wide range of visual programming systems was complete, a scenario-based analysis of two

environments was conducted (Carroll, 2000; Rosson & Carroll, 2002). In the final step of Phase I, empirical evaluations were used to examine the usability of several end-user programming systems, focusing on the issues identified through the usability inspections and scenario-based analysis (see <http://csgrad.cs.vt.edu/~seals/research/Agents> for more details). This resulted in an initial set of requirements for the iterative design and development work conducted in Phase II.

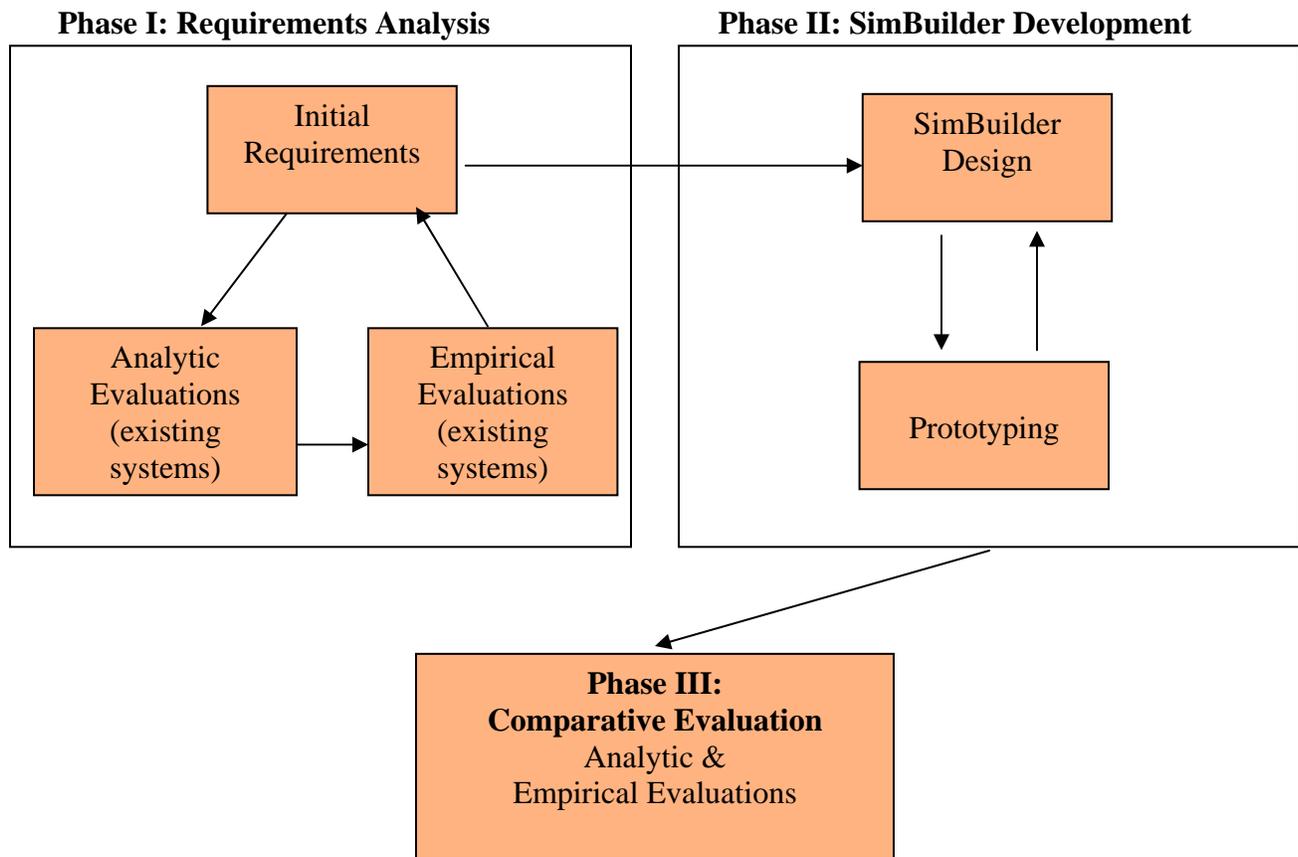


FIGURE 3.1 RESEARCH APPROACH

In Phase II, the SimBuilder development cycle, the design was refined, a prototype was created, and early stages of the prototype were iterated. In Phase III, a comparative evaluation (consisting of both analytic and empirical analyses) assessed the success of the design by contrasting teachers' learning and use of SimBuilder versus a state-of-the-art alternative, AgentSheets. Phase III included a comparative usability inspection of AgentSheets and SimBuilder, as well as a detailed empirical study using

both qualitative and quantitative outcome measures. The three phases were designed to answer the research questions discussed in the following section.

3.3 Research Questions

The first phase of the research was exploratory, intended to analyze the challenges faced by teachers trying to use visual simulation environments, as well as to explore the features and user interfaces offered by a variety of visual programming tools. As detailed in Chapter 4, these exploratory analyses led us to focus on the three general problem areas of learning, programming style, and reuse.

The study addressed the following research questions:

1. *What are the key learning challenges for teachers working with visual simulation environments?*

1a. *Programming Style:* State-of-the-art systems like AgentSheets and Stagecast Creator rely on graphical rewrite rules in which the details of the visual specifications are crucial to the success of programming (Phase I, feature comparisons and claims analysis). Requirements analysis (Phase I) suggested that the graphical rewrite technique produces “brittleness” in the programming paradigm that may be problematic for novices. The design of SimBuilder responded to this issue by providing a more flexible visual construction and specification medium. Subsequent empirical studies examined the effectiveness of the new design created in Phase II, using the comparative evaluation in Phase III.

1b. *Tool Complexity:* Existing examples of visual simulation environments rely extensively on direct manipulation techniques for creating, editing, and testing simulation components (Phase I feature comparisons and claims analysis). One consequence is that users must often open and work with a multitude of small windows, icons, buttons, etc. Requirements analysis (Phase I; see Chapter 4 for

details) suggests that this highly intensive “tool-busy” environment may be problematic for novices. SimBuilder offers a simplified operating environment, whose consequences are assessed in the final phase of empirical analysis.

2. *How can we assist teachers in learning and performing visual simulation programming?*

2a. Programming Style: Prior work on object-oriented programming has claimed that objects are a “natural” way to decompose real world problems into computational solutions (Rosson & Alpert, 1990). In this study, an object-oriented simulation framework is used to address some of the problems with current rule-based visual simulation environments because it enables users to adopt a problem-oriented conceptual focus instead of a visual animation focus. The empirical comparison of SimBuilder and AgentSheets (Phase III) was used to gather evidence on this issue.

2b. Reuse: Some of the most tedious aspects of visual programming involve the initial creation of components and simple behaviors, such as horizontal and vertical movement. This delays learning and creates motivational obstacles for continued use. Offering teachers suitable material for reuse in their programming is expected to speed and motivate their learning. The requirements analysis study of AgentSheets (Phase I; see Chapter 4 for details) suggested that teachers were able and willing to reuse material. The more extensive design work in Phase II and the comparison of SimBuilder and AgentSheets in Phase III was used to further investigate the reuse behaviors of these end-users.

2c. Expressivity: Expressivity is a highly qualitative measure and, as defined by Merriam Webster (www.m-w.com), is the quality of being expressive, conveying a meaning or feeling. In simulation design, expressivity alludes to the ability of a designer’s creation to realistically express the basis of a real world model. This study relates specifically to objective-subjective expression (Britannica online, www.britannica.com 2004). The form that is used, i.e. the environment, affects the

structure of what is being created by the user. Based on the structure imposed by the environment, the expressive quality of the artifacts and the form imposed by their environments are judged, thus making an objective-subjective qualitative evaluation of the user created artifacts.

3. *What forms of reusable material are suitable for teachers building simple science simulations?*

3a. Generic vs. Examples: Humans learn well from examples. However, it is not clear at what level of granularity such examples are most useful. Requirements analysis (Phase I; see Chapter 4 for details), for example, suggests that teachers are better able to reuse generic components (e.g. a “mover”) than a piece of a concrete world (e.g. a “waste chemical molecule”). The design of SimBuilder included provision for reuse of both generic component, or template, examples and concrete examples, and the final empirical studies again examined the effectiveness of these two forms of both SimBuilder and AgentSheets (Phase III).

In addition to identifying these three general research questions, the exploratory work of Phase I was used to motivate several specific hypotheses associated with the three research questions and these hypotheses were tested in the final empirical study that compared SimBuilder with AgentSheets. In general, data collected in support of the Phase III hypothesis testing included data such as performance times, analysis of the simulations produced (including expert ratings thereof), and participants’ ratings of the simulation tools, materials, and programming experience. Interpretation of these data was further enriched through qualitative observations collected during the experimental sessions and interviews with participants afterwards. The hypotheses tested in Phase III are included in the following section, and each hypothesis ends with a link to the relevant section of the associated data table.

Hypothesis One: Learning

The object-oriented SimBuilder will be easier to learn, and viewed as being more usable by novices than the control environment AgentSheets, which uses graphical rewrite rules.

H₀: There will be no significant difference between the usability and learnability of the control and experimental environments in terms of user ratings of overall satisfaction, ease of use, motivation to use, and fun using the environment.

H_A: There will be a significant improvement in teachers' overall satisfaction with SimBuilder vs. AgentSheets (Table 6.24).

H_B: There will be a significant improvement in teacher's satisfaction with the ease of use, motivation to use, and fun in using SimBuilder vs. AgentSheets (Table 6.26).

H_C: There will be a significant decrease in teachers' learning time for SimBuilder vs. AgentSheets (Table 6.20).

H_D: There will be a significant increase in the number of objects and rules created by teachers using SimBuilder vs. AgentSheets during the learning phase of the comparative study (Table. 6.21).

Hypothesis Two: Programming Style

The object-oriented message-passing programming style incorporated in the SimBuilder environment will facilitate simulation construction relative to the animation-oriented graphical rewrite rule programming style of AgentSheets.

H2₀: There will be no significant difference in ease of user interaction with the animation-oriented graphical rewrite rule programming style of the control environment AgentSheets and the object-oriented message-passing programming style incorporated in the experimental environment SimBuilder.

H2_A: There will be a significant difference in teachers' satisfaction indicating good support for novice programming (i.e. easy to remember tool location, easy object and rule creation, and increased user confidence that they can create a working simulation in rating SimBuilder vs. AgentSheets (Tables 6.25 and 6.27).

H2_B: There will be a significant improvement in the complexity measures of teacher created artifacts in favor of SimBuilder vs. AgentSheets programming (Table 6.21)

The empirical studies of Phase I have suggested that reusable material that is relatively generic and abstract (e.g., a starter kit of components) are more readily understood and reused than concrete example simulations.

Hypothesis 3: Reuse

Generic components will facilitate simulation construction more effectively than concrete example simulations.

H3₀: There will be no significant difference between SimBuilder and AgentSheets in user reported understanding of generic components vs. example components when used to build new simulations.

H3_A: There will be a significant improvement in SimBuilder over AgentSheets in user-reported understanding of generic components vs. example components based upon their simulation creation experience (Tables 6.28).

H3_B: There will be a significant increase in numbers and/or complexity of user created artifacts when using generic components vs. example based components reused to build new simulation (Table 6.23).

Hypothesis 4: Expressivity

Objects and simulation worlds created in SimBuilder will be more sophisticated and have a more aesthetically pleasing and natural look than those created using the control environment.

H4₀: There will be no significant difference between the control environment and SimBuilder in user interface experts' evaluation of teacher created artifacts.

H4_A: There will be a significant improvement in expert ratings of visual quality and action quality of user created artifacts in the SimBuilder over the control environment (Table 6.24).

This chapter introduced the research problem, associated questions, and hypotheses addressed by this study and described the characteristics of the analytic and empirical research context. This research will explore these research questions and hypotheses, and support the further study of reuse techniques in novice end-user programming. Details of the analytic and empirical methods used for requirements analysis appear in later chapters, including both the methodology for the study of learning and reuse in AgentSheets (Section 5.2) and the methodology for the learning study of Stagecast Creator (Section 5.4). The methodology for summative evaluation as a comparative evaluation of AgentSheets and SimBuilder will be discussed in Section 6.2.

CHAPTER 4. PHASE I: REQUIREMENTS ANALYSIS

This study's requirements analysis phase encompassed many research activities, highlighted in Figure 4.1. The first was a study of end-user construction environments which was used to create a taxonomy of tools, illustrated in Table 4.1. Next a more detailed, yet informal, usability inspection of five visual end-user programming environments was conducted (Table 4.2). This was followed by an analytic evaluation of two environments, AgentSheets and Stagecast Creator, in the form of a scenario and claims analysis in an effort to clearly define the requirements for a robust end-user programming system. AgentSheets was selected for an empirical pilot study of learning & reuse and an empirical study of learning in Stagecast Creator was performed in order to discover more about novice programmers and their direct manipulation techniques. To further refine the requirements for the final study and become more familiar with their domain of discourse about computers and programming, a longitudinal guided exploration of Stagecast Creator was carried out with the assistance of two middle school science teachers. This chapter concludes by discussing the implications for educational simulations for teachers and, as the final activity of the requirements analysis, describes the set of initial system requirements utilized to refine subsequent studies and to begin SimBuilder development.

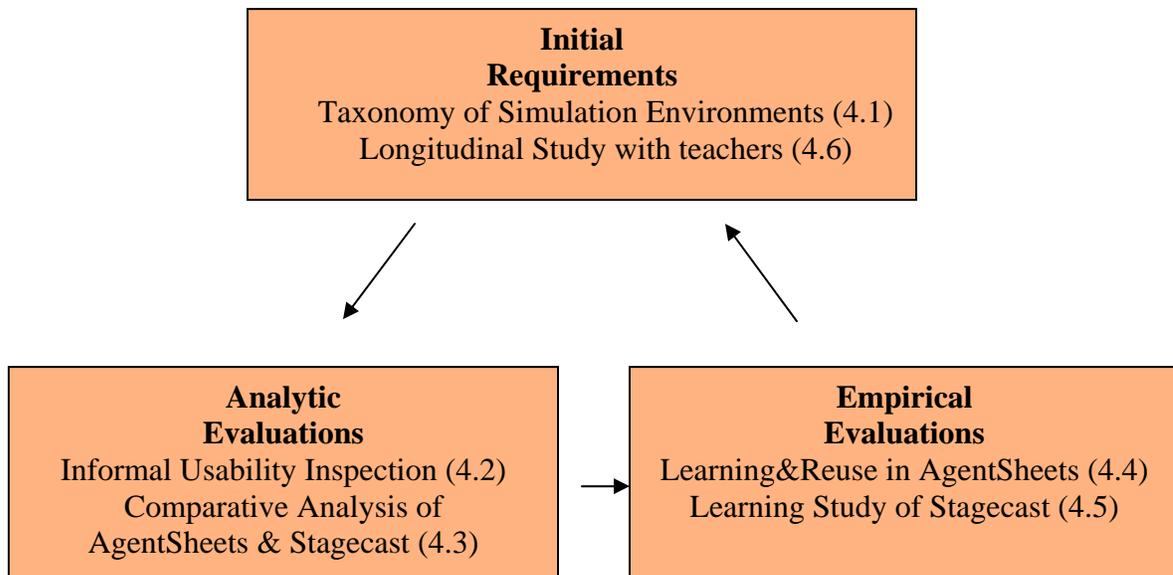


FIGURE 4.1. PHASE I REQUIREMENTS ANALYSIS

4.1 Taxonomy of Simulation Environments

The preliminary stages of this research project defined the goals of providing software with improved usability in order to increase teacher adoption of new educational software technology in their classrooms. Existing simulation creation software was studied in order to select software that most closely matched teachers' classroom needs, and then modified or new software designed and created to better support teachers and their reuse strategies. The first step was to conduct a language survey and create taxonomy to narrow the field of candidates for further study. The criterion for selecting these languages was that they should represent different types of software that will support educational simulation. At least one example from each of the following categories was evaluated: visual programming, programming by demonstration, simulation/construction kits and software reuse/visual OOP. Table 4.1 summarizes these categories and the candidate languages within each.

The visual programming environments category is based on the use of completely visual environments such as LabVIEW, a scientist's construction kit used to build virtual instruments by direct manipulation. The system programming is handled by three palettes, which are used for coding, debugging, and creating. All of the programming is handled by icons in these palettes, but the wealth of icons in this environment would be very complicated for novice users who must negotiate the cognitive overload of learning to create programs with sequences of icons as in traditional programming with sequences of text. LabVIEW would be the most useful in this category for building working educational simulations, but this selection is very complex and also cost prohibitive (i.e. greater than \$1000) for the average middle science schoolteacher.

Table 4.1.

Abbreviated Taxonomy of Languages Studied

(See Appendix A for full list with descriptions)

Language Type	Language
Visual Programming Environments	LabVIEW
	Tinkertoy
	PICT
Programming by Demonstration	Pygmalion
	Tinker
	Rehearsal World
	Cocoa
	Mondrian
	Stagecast Creator
Simulations/Construction Kits	Garnet (built from Peridot)
	Amulet (built from Garnet)
	ActivChemistry
	AgentSheets
	Geometer's Sketchpad
	HyperStudio (built from HyperCard)
	Pinball Construction Set
	SimCity
Star Wars Droid Works	
Software Reuse/Visual Object-Oriented Programming	PROGRAPH
	HI-VISUAL
	ThingLab

The programming by demonstration category uses environments where all the rules are created with the use of a macro recorder device and the demonstrations of actions are rendered as rules. This is acceptable for simple rules, but makes the creation of more complex rules more difficult. For example, Stagecast Creator is a simulation micro-world building environment where programming actions are created with programming by demonstration techniques, and rules can be modified with direct manipulation. This environment is geared to support novice programmer usability and

would be useful in the creation of educational simulations. It can be procured for minimal cost (i.e. less than \$100). ActivChemistry, a chemistry construction kit that provides a fixed number of parts to combine to perform experiments, was also a very good candidate for further study, but only supports the Chemistry content area. This study aims to identify software packages that can be used by all science and mathematics teachers, so the environments studied must be able to provide more generalized solutions.

In the simulation/construction kit category, environments support the construction kit style of rule creation. The user simply chooses the rules they would like to use and assigns an order for rules to fire. This is a very good technique for novice users attempting to create educational simulations and AgentSheets had already been used by students to create over a hundred simulations. It can also be procured with minimal cost (i.e. less than \$100).

In the software reuse/visual object-oriented programming category, environments are object-oriented in their implementation, and support software reuse. These environments would be very good for programmers to build simulations, but offer little support for novice programmer simulation creation.

Creating this taxonomy of visual languages reduced the field of candidate languages to those that would be easily accessible to teachers and not too cost prohibitive. Languages were also expected to support novice learning and reuse in the creation of visual educational simulations. Thus, three simulation/construction kits (AgentSheets, Cocoa, and HyperStudio) and one programming by demonstration system (Stagecast Creator) were selected for use in this study.

4.2 Informal Usability Inspections

In a more detailed analytic analysis, an informal usability inspection of the four visual end-user programming environments selected was completed, as shown in Tables

4.2, 4.3, and 4.4. A usability inspection provides information that helps identify potential usability issues. This effort supplied more information to use in isolating which systems would be good candidates for further analytic evaluation and future empirical evaluation.

The usability inspection consisted of three sections that dealt with the following: environment issues, drawing tools, and rules. A two-person team who were very familiar with the functionality available in these environments performed the inspection and worked to reach a consensus on ratings.

Table 4.2.

Informal Usability Inspection of Four Systems (Environment Issues)

	AgentSheets	Cocoa	HyperStudio	Stagecast
Standard components (e.g. condition/actions)	1	2	1	2
Rule-based object behaviors	1	2	3	2
Realistic animation of objects	2	2	3	2
Incremental testing	1	1	3	1
Object copying	1	1	1	1
Window management	3	2	1	2
Library of simulations	1	2	2	1
Web accessibility and use	1	1	1	1
Global variables for state change	1	2	3	2
Environment start-up	2	2	1	1
Cross-platform use	2	3	2	2
Reuse-adaptation of objects	2	2	1	1
Auditory feedback for actions	2	3	3	2
Default new object	3	3	1	1
Visual editing of rules	2	2	3	2
Simulation speed controls	3	2	3	2
Object destruction	3	3	1	1
Saving projects or worlds	1	1	1	1
Icon interpretability	2	2	2	2
Multiple active projects	3	3	2	1
Support classes of similar objects	3	3	3	1

Usability Inspection Levels: 1. Very good 2. Satisfactory 3. Unsatisfactory

The first part of the inspection identified environment issues to determine which system showed the most promise for building educational simulations by novice programmers (Table 4.2). Environmental features were ranked according to their importance to usability and those criteria used to choose an environment for more detailed study. Of the items that received the highest priority, such as standard components, rule-based object behaviors, realistic animation of objects, incremental testing, object copying, window management, library of simulations, and web accessibility, the systems AgentSheets and Stagecast Creator were rated *very good* in all categories except window management, where they rated *satisfactory*. HyperStudio was rated *very high* in all our top priority issues except visual editing of rules and rule-based object behaviors, where it rated *unsatisfactory*.

The environment issues assigned the second level of priority included the following: global variables for state change, environment start up, cross platform use, reuse adaptation of objects, and auditory feedback for actions. AgentSheets was rated *very good* in global variables for state change and *satisfactory* in cross platform use and reuse adaptation. Stagecast was rated *very good* in reuse-adaptation of objects, and *satisfactory* in realistic animation and global variables. HyperStudio ranked *unsatisfactory* for global variables and auditory feedback. The results from the environment usability inspection thus favored the selection of AgentSheets and Stagecast Creator for the next phase.

The second part of the inspection identified issues dealing with functionality provided by drawing tools to determine which system would best support drawing by our participants (Table 4.3). The items assigned the highest precedence were easy undo, flexible point size and flexible object size. Stagecast Creator supported all of the issues except for a robust undo function. AgentSheets was *unsatisfactory* in all categories.

Table 4.3.

Informal Usability Inspection of Four Systems (Drawing Tools)

	AgentSheets	Cocoa	HyperStudio	Stagecast
Easy undo	3	3	1	1
Flexible point size	3	1	1	1
Flexible object size	3	3	2	2
Paint fill (e.g. paint bucket)	2	2	1	2
Predefined drawing objects	3	3	1	2
Drawing canvas reset	2	1	1	1
Imported images	2	2	2	2
Desktop color sampler	2	2	3	2
Block object fill	1	3	3	3

Usability Inspection Levels: 1. Very good 2. Satisfactory 3. Unsatisfactory

The third part of the inspection identified issues dealing with functionality provided by rule creation facilities to determine which system would best support building of working educational simulations by users (Table 4.4). The items assigned the highest priority were ease of rule creation and rule priorities. AgentSheets had the best support for ease of rule creation and rule priorities, with a rating of *satisfactory*.

Table 4.4.

Informal Usability Inspection of Four Systems (Rule Creation)

	AgentSheets	Cocoa	HyperStudio	Stagecast
Ease of rule creation	2	3	NA	3
Rule priorities	2	3	NA	3
System extensibility	2	NA	NA	NA
Programming by demonstration	NA	2	2	1
Rule activation tracing	2	2	3	1
Graphical rewrite rules	1	2	NA	2

Usability Inspection Levels: 1. Very good 2. Satisfactory 3. Unsatisfactory NA. Not Applicable.

AgentSheets provided an overall satisfactory environment, although using less than optimum drawing tools, but its choice is justified because it offers much more flexibility than any of the other environments in terms of rule creation. This is very

important in creating meaningful working simulations. Stagecast Creator provided a satisfactory creation environment, ease of use in creating rules, and robust drawing tools. HyperStudio was the best in terms of drawing tools and has some satisfactory environment issues, but did not support flexible rule creation. Based on this informal usability inspection, the programming construction kit AgentSheets and the programming by demonstration system Stagecast Creator were selected as the best candidates for more formal analytic study.

4.3 Analytic Comparison of AgentSheets and Stagecast

The second step of Phase I, the requirements analysis, performed a comparative analysis of AgentSheets and Stagecast involving an analytic comparative evaluation in two forms: a feature comparison, and a scenario and claims analysis. Both techniques were used to refine the requirements for the design of the proposed system. This section gives an overview of the two systems and performs a feature comparison, followed by a scenario and claims analysis. The section ends with a summary and conclusions drawn from this analytic work.

4.3.1 System Overview

AgentSheets is a substrate for building domain-oriented, visual, dynamic programming environments that do not require traditional programming skills. It features a versatile construction paradigm to build simulation components and configurations for a wide range of problem domains, such as art, artificial life, distributed artificial intelligence, education, environmental design and computer science theory (Figure 4.2 shows a sample simulation). Agents can exercise different communication modalities, such as animation, sound, and speech, as well as visual communication through color, shape, and movement (Repenning, 1993).

AgentSheets is based on the use of rewrite rules; users create simulations by drawing agents for a microworld and then add behavior by adding rules that can be created through either direct manipulation or by using its construction tool kit. A basic

rule is created by dragging a blue condition into the *if* portion of the rule palette and dragging a red condition into the *then* portion of the rule palette. After the rule is created, the user selects “apply” to accept this new rule.

Consider the case of a simple example rule, which might be “*if* you are an auto, *then* you move to the right.” In AgentSheets, this looks like:



Figure 4.3a shows a more complex example, where the agent *Sunny* creates *Sunrays*. Figure 4.3b illustrates how the behavior of sunrays is regulated to allow them to move in a psuedo-random fashion. As a final step, an AgentSheets simulation can be transformed into Java applets, with the Ristretto option. This allows users to store and share their creations in a web-based repository (<http://www.AgentSheets.com/client/http/agents/index.html>).

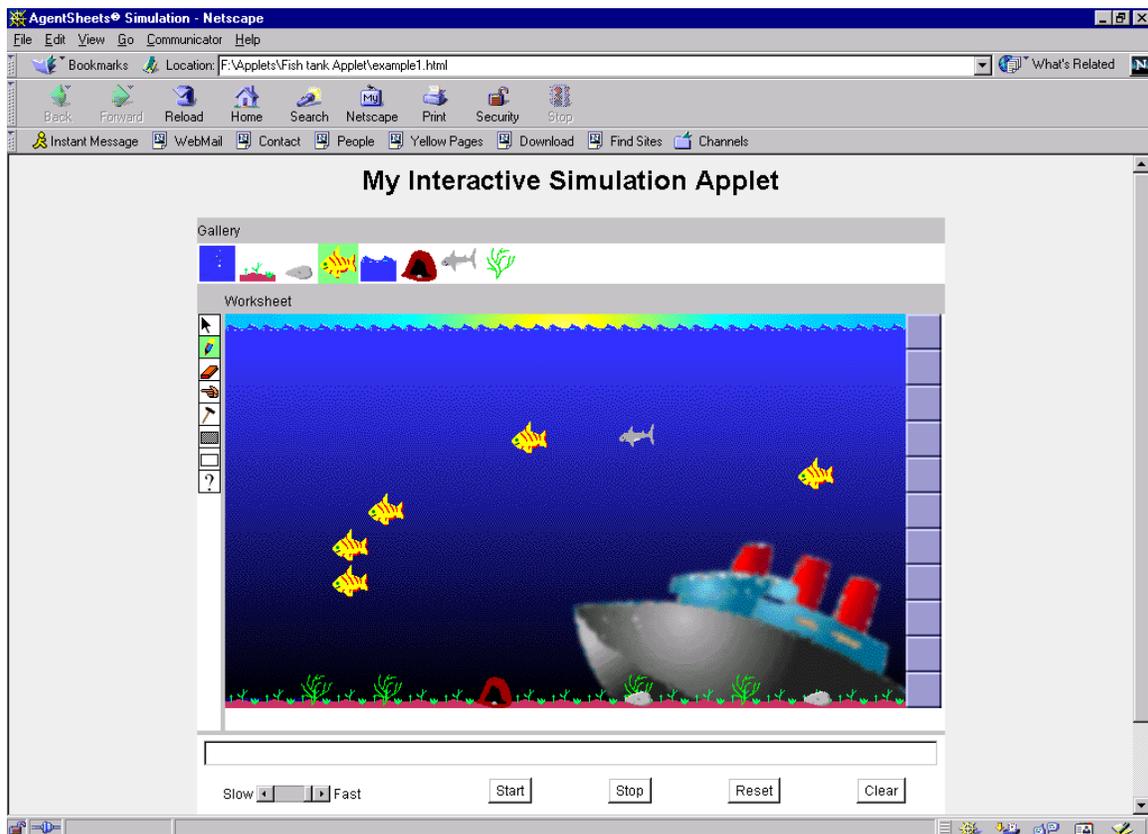
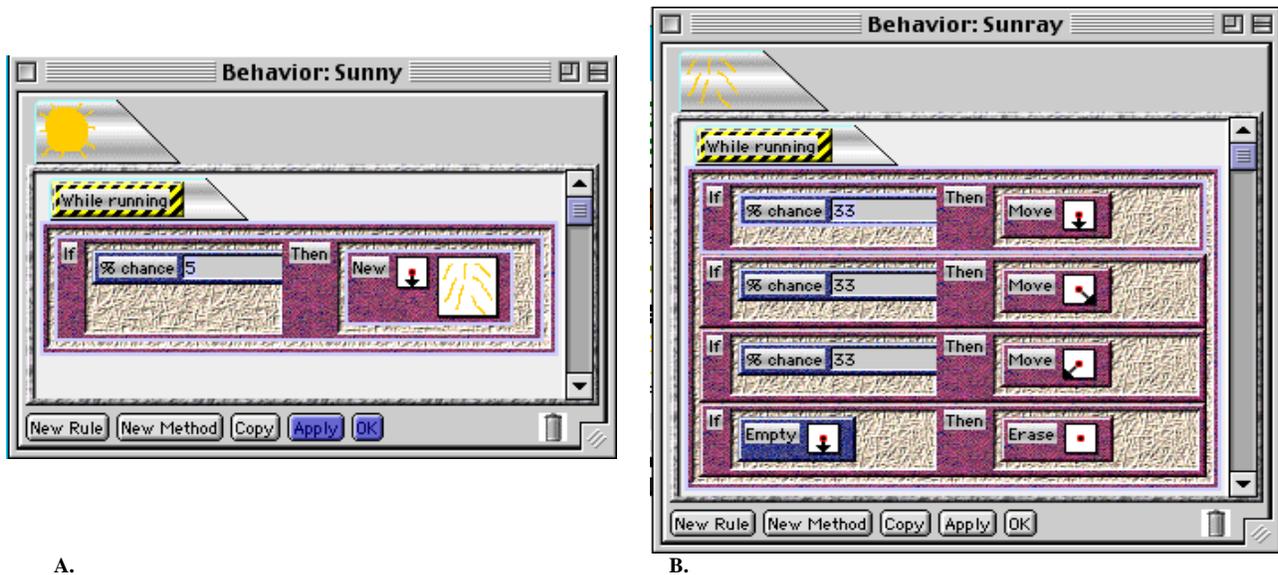


FIGURE 4.2. AGENTSHEETS BEHAVIOR PALETTE (I)



A.

B.

FIGURE 4.3. AGENTSHEETS BEHAVIOR PALETTES (II)

AgentSheets approaches programming as problem solving, with a construction paradigm that helps users to incrementally create and modify representations. With many “one-size fits all” programming applications, users only have access to general-purpose tools that may not serve their needs. With a substrate such as AgentSheets, novice developers are able to craft, manipulate, and reuse tools to meet their specific needs. In AgentSheets, metaphors serve as mediators between problem solving-oriented construction paradigms and domain-oriented applications. “Metaphors are used to represent application semantics by helping people to conceptualize problems in terms of concrete notions. Furthermore, metaphors can simplify the implementation of applications. Application designers can explore and reuse existing applications that include similar metaphors” (Repenning, 1993).

Students, teachers and developers have used AgentSheets to create many applications for different domains, both independently and collaboratively. For example, a group of 4th and 5th grade students built simulations of ecosystems (Rader et al., 1997; Lewis et al., 1998; Cherry, Ioannidou, Rader, Brand & Repenning, 1999). A doctor built an environment to illustrate the effect of a drug on his patients (Cherry et. al., 1999; Repenning, 2000). Other groups have built simulations of artificial life environments, and

simulations for physical, biological, mathematical, earth and social sciences (Cherry et al., 1999).

AgentSheets has a behavior processor, created by analogy to the word processors that allow end-users to manipulate static documents at a high level. A behavior processor enables end-users to manipulate interactive content such as simulations and animations. Building a simulation forces users to go beyond TV-style screen design. Users must transform information into knowledge in the process of mapping that information onto behaving agents in a simulation.

It is highly desirable for end-users to modify or enhance domain specific applications, and mechanisms are needed that provide this flexibly without forcing end-users to become programmers. Experts use Object Oriented (OO) techniques to locate promising class hierarchies and refine them by adding or extending methods to provide the new desired behavior. In reuse, most novice users resort to a copy and paste approach rather than making appropriate use of OO techniques:

Object-oriented concepts relying on single and multiple inheritance can be too complex for end-users, and non-inheritance approaches too limited. Semantically enriched rewrite rules avoid problems caused by overgeneralization and non-visual Abstractions. AgentSheets utilizes Graphical Rewrite Rule Analogies (GRRA)s to allow reuse without the oversimplification, overgeneralization and unnecessary abstraction pitfalls of other approaches (Perrone & Repenning, 1998).

These visual analogies are good reuse mechanisms for end-users who otherwise would have resorted to copy and paste (e.g., cars move on roads like trains move on tracks). End-users tend to think concretely rather than abstractly. Inheritance works for professionals trained in abstraction, but fails to work for end-users. End-users are generally successful with HyperCard, where reuse consists of only copy and paste.

GRRAs can be used to provide alternative programming solutions, which facilitate reuse for novices and avoid the complex pitfalls of an OO approach.



FIGURE 4.4. STAGECAST CREATOR FLOWER GARDEN

Stagecast Creator is a simulation toolkit for both Microsoft Windows and Macintosh that enable users to easily create interactive stories, puzzles, games, and simulation models (Figure 4.4). It is the successor to Cocoa, a children's sprite-world creation kit developed at Apple by David Smith and Allen Cypher using Sk8 initially and later using PROGRAPH. Stagecast can be used by novices to create simulations. It has a comprehensive set of drawing tools, and the user can create simple simulations through direct manipulation. With Stagecast, novice developers can craft simulations that meet their specific needs.

An assessment of some of the key functionalities of AgentSheets and Stagecast is shown in Table 4.2. For more details, see Appendix B. Both environments utilize GRRs

to handle complex situations and equations. Variables can be easily defined in either environment, but in Stagecast functionality of the variables is limited to predefined functions available within the systems construction kit. With variable naming, Stagecast is much more malleable than AgentSheets. If users want to reuse a character, they can either copy it or simply change its name. In AgentSheets, users are allowed to copy and clone agents, but the agents' functionalities remain the same as their base class. The Stagecast environment has a form of end-user "training wheels", and hides some of its functionality in drawers that can be opened and used by developers or creators who wish to do more than just run simulations.

4.3.2 AgentSheets/Stagecast Feature Comparison

In both AgentSheets and Stagecast, basic user functions are predefined. An additional feature of AgentSheets allows sophisticated users to extend the construction kit with user defined functions. The analysis determined that AgentSheets' basic construction kit allowed users more flexibility and a larger variety of rule behaviors than Stagecast. This flexibility gives users added power to create more realistic simulations. Three palettes in AgentSheets, the condition, action, and triggers palettes, allow novices to create simulations by using direct manipulation techniques. Stagecast has a basic rule construction kit, and also allows users to create rules easily by demonstration through a macro-recording device. Stagecast has a more game-like facility; it can accommodate multiple stages (e.g., a hard stage and an easy stage) of a game or simulation easily with a door between stages. Stages can be traversed with a door or a stage button. Creator connections are inherently tied to their visual representations and the characters' internal or global variables (for more details of Evaluation of Stagecast see Appendix C).

The two environments share one important characteristic: the production-system model of condition-action pairs allows condition testing to occur in an ordered fashion. This model introduces many subtle complexities, perhaps the most difficult thing to debug. Both environments provide some debugging facilities (e.g., the ability to block one or more rules in a set in Stagecast, or the ability to test all or part of a behavior on a particular character in AgentSheets).

Both environments show a great deal of promise in the area of end-user programming of interactive, dynamic simulations. The Stagecast environment is more visually appealing, and there is less cognitive overhead prior to beginning to build a simulation. Users need only minimal knowledge, and they do not need to know anything about programming. They can begin creating a character immediately, and then give it actions that help it to interact within the microworld in which it was created. Using the rule tool, and programming by demonstration and direct manipulation, they can add the character's actions.

Once users have added more characters to the microworld, it becomes computationally expensive to match many different scenarios. For basic simulations this is not problematic. Stagecast has incorporated a tool to reduce the number of situations for which there must be an exact match by using the "Don't Care" condition. The "Don't Care" enables basic simulations to ignore secondary characters and only evaluate responses to the actions of the main character in each scenario. The presence and actions of surrounding characters do not stop the action. It is a further advantage that Stagecast will run on multiple platforms, including both Mac and Windows.

It is also easy for end-users to start creating an agent and adding behaviors in the AgentSheets environment. In many cases, it functions better than Stagecast in creating more involved simulations because of its expanded rule facility (i.e., Condition and Action Palettes). Some problems in the environment need to be worked out to make it more visually appealing, with improved drawing facilities. Efforts need to be made to reduce the complexity encountered by users when creating complicated simulations relying on variables and when making slight variations in simulation execution. The robust set of rules currently available in AgentSheets makes it a tool that will be more suited to the complex interaction of a detailed simulation.

During feature exploration, positive and negative features were identified for both environments that will impact usability, along with some low-level components that can

be reused. Further studies will explore the reuse of higher-level components and simulations will be conducted to enhance the support system for an on-line teacher community. The next step in this analytic analysis is to perform a scenario and claims analysis.

4.3.3 AgentSheets vs. Stagecast: Scenarios and Claims

The initial evaluation also used scenarios and claims analysis. In prior work, scenarios and claims analysis were used to perform analytic evaluations of a variety of interactive systems (Carroll & Rosson, 1990; 1992; 1995; Rosson & Carroll, 1993; 1995). A user interaction scenario narrates user goals, expectations, and reactions that are part of an episode of use, as well as a user's concrete interactions with the system. Although the features of a system can be analyzed in isolation (e.g., by simply listing them), weaving a feature into a usage scenario provides a realistic context for reasoning about its impact on use (Carroll, 1995).

A claim documents the possible consequences of a feature in a usage situation. Some consequences are positive (the upsides); some are negative (the downsides). For example, a blinking icon can attract users' attention to an important task element, but it can also distract them from work on some other element. The design judgment about whether to use a blinking icon depends on the usage situation and on how important gaining the users' attention is relative to the cost of annoying or distracting them.

The strength of claims analysis is that it promotes a balanced analysis of design features. Designers typically focus on the upsides of a feature (i.e., why they included it), while usability specialists tend to see the downsides (i.e., the usage problems a feature causes). Claims analysis assumes that identifying and rationalizing tradeoffs is a fundamental element of design. Ultimately, designers must take positions (explicitly or not) on a variety of tradeoffs. Claims can be used to record both sides of an argument.

As an analytic evaluation, claims analysis can be used directly to guide design: the designer attempts to maintain or enhance upsides while removing or mitigating

downsides (Carroll & Rosson, 1992). Thus, in efforts to design new visual programming environments, a claims analysis of existing environments can articulate the strengths and weaknesses of individual features. However, claims analysis also supports which terms mediated evaluation (Scriven, 1967): claims serve as hypotheses that direct the collection of data (e.g., guiding the construction of tasks that exercise the features of interest), as well as help to interpret the usage data collected (e.g., the successes or problems observed for a feature compared to the hypothesized consequences).

Two scenarios were developed and elaborated to examine the usability of the environment with respect to both immediate and extended use. In preparing to create any tool designed to support model construction, the mapping between the user's mental model and the systems capabilities is critical. The user's image of reuse has major implications for the sustainability of a learning community of teachers. The scenarios were as follows:

Scenario 1: Exploration and Learning (Environment Focus)

Scenario 2: Adaptation of Existing Projects (Reuse Focus).

Note that neither the tutorial nor other documentation were analyzed. Even though this has been an important design component of the Stagecast package (Cypher, personal communication, 2000), it has been less of a focus for the AgentSheets package. Stagecast includes examples and a tutorial with the environment. AgentSheets has been developed largely as a research project, and has only recently begun to emphasize documentation. Some of the examples and tutorials created during this study are also included on the AgentSheets Distribution CD.

For illustrative purposes, claims analyzed Scenario 2 is examined here in greater detail. The scenario is as follows:

A teacher wants to build a model of an ocean food cycle for her 5th grade class. The model would include underwater vegetation, which grows as result of receiving sunlight. The model contains fish that swim throughout it. Small fish

survive by eating the vegetation and larger fish survive by eating the smaller fish. Fish that don't eat will die, and fish that eat will grow. Also, if pollution is introduced into the model it will cause some fish and vegetation to become sick and die.

To begin the reuse session the teacher searches for a similar simulation that she can reuse to build a modified ocean food cycle. When she finds a suitable model fish world, she runs the simulation to find out what the model does. She identifies agents and behaviors to access, which will be useful for her modified simulation. She remembers how to create new agents from her recent learning tutorial and begins to add necessary agents that don't exist in this model. (For example, she creates a new seaweed agent). She remembers that she has seen the Sun agent before in the water cycle model and decides to copy and reuse the Sun, whose Sunrays will cause the Seaweed to grow. She would also like to reuse the behavior of flowers growing in the Flower Garden Model to give her seaweed a life cycle. She keeps the basic fish agents, but decides to add a new fish agent, Whale, that will eat the Shark, which currently eats Goldfish. She does this by getting a Whale from clip art or by copying Shark and making it larger. She is also able to reuse the behavior of Shark by analogy. The Whale eats a Shark, just like a Shark eats a Goldfish.

Reuse Claims for AgentSheets

Details of the associated claims for AgentSheets are listed in Table 4.5. It is not easy for users to reuse the example because only one world can be active at a time. This may have been intentional on the part of the designer in order to keep novice users from becoming confused due to multiple windows being open simultaneously. However, this design of allowing only one active world makes novice reuse harder, by limiting the copy-paste reuse technique. If users could open more worlds, they would be able to learn and reuse by example more readily.

Table 4.5.

Claims Analysis of a Reuse Scenario: AgentSheets

Usage Feature	Possible Upsides (+) or Downsides (-) of Feature
Allowing only one world to be active at the same time	<ul style="list-style-type: none"> + prevents the user from getting confused with too many windows - BUT makes reuse harder. The user can reuse other agents and simulations from the behavior exchange, but not within the AS environment
Supporting the renaming of agents	<ul style="list-style-type: none"> + enables a copy (clone) approach to adaptation and reuse of agents - BUT users cannot adapt clones' behavior, which is directly inherited from the base object that was cloned
Multiple depictions	<ul style="list-style-type: none"> + allows the user to have several related agents of the same class or type - BUT the appearance is the only thing that can be modified. Each new instance will share or directly inherit the same rule set
Analogy editor	<ul style="list-style-type: none"> + allows the user to reuse behavior analogy (e.g. a plant absorbs sunlight like a bigger fish eats a smaller fish)
Rules for classes	<ul style="list-style-type: none"> + all classes of fish could have the same behaviors - BUT this would restrict the fish to having identical behaviors

Currently there are only two methods of reuse in AgentSheets. One is cloning of an agent, which is an exact copy of the agent's look and behavior. Another is importing an agent through the Behavior exchange. The system supports multiple looks or depictions, which allows users to modify the agent's look. Unfortunately, users cannot adapt the clones' behaviors. Studies have been conducted with an AgentSheets' analogy editor (Perrone & Repenning, 1998), which allows users to reuse behavior by analogy (e.g., a car runs on the road, like a train runs on a track).

Another very useful claim is that AgentSheets supports rules for classes. For example, all classes of fish can have the same behaviors. However, this does require that all fish have identical behaviors.

Reuse Claims for Stagecast

Details of the associated claims for Stagecast are listed in Table 4.6. Users are able to have two worlds open at the same time, which is helpful for reuse but may be confusing since users will have to keep up with characters and rules for the two worlds. It does support the renaming of agents by enabling either a copy-edit (mutation) approach to adaptation or the reuse of characters. This can be very helpful when reusing characters that have a similar look or behavior. The system supports multiple depictions, supports easy changes between depictions with mouse selection, and allows the modification of appearance so that each new instance can have a distinct variable state that is not restricted to being a global variable. Stagecast, like AgentSheets, supports rules for classes, referred to in Stagecast as jars (e.g., all fishes that have the same behavior are placed in the same jar).

After completing an initial analytic analysis of Stagecast and AgentSheets, some initial findings included the advantages and disadvantages of both environments.

Table 4.6.

Claims Analysis of a Reuse Scenario: Stagecast

Usage Feature	Possible Upsides (+) or Downsides (-) of Feature
Allowing more than one world to be active at the same time	<ul style="list-style-type: none"> + simplifies copy and edit of prior characters in building a new world - BUT keeping track of characters and rules for the two worlds can be difficult
Supporting the renaming of agents	<ul style="list-style-type: none"> + enables a copy-edit (mutation) approach to adaptation and reuse of characters - BUT may be tempted to adapt characters that are quite dissimilar, causing more work for themselves
Multiple depictions	<ul style="list-style-type: none"> + allows the user to have several related agents of the same class or type + user can easily change between multiple depictions with mouse selection + appearance can be modified for each new instance and each can have a distinct variable state - BUT the appearance is the only thing that can be modified. Each new instance will share or directly inherit the same rule set
Rules for classes	<ul style="list-style-type: none"> + all classes of character can have the same behaviors, but there is no restriction on modifying behavior to exhibit individual differences

4.3.4 Implications and Conclusions

After analytic analysis the feature comparisons and scenario analysis of AgentSheets and Stagecast revealed that the two environments are best suited for different purposes. Stagecast is preferable for very simple demos (such as might be created by younger students). At an intermediate level, AgentSheets seems more sophisticated because it offers more options for conditions/actions, and has an environment very conducive to exploration and reuse at various levels. As the model becomes more complex, Stagecast uses literal stages to compartmentalize logical stages (i.e., states) of the model's execution. For special-purpose modeling, AgentSheets gives

ultimate flexibility because its condition and action palettes can be extended. These palettes are a customizable part of the environment. Experienced system users can create new primitives to extend the existing set.

Some other concerns addressed through claims analysis are the following (see Appendix B for an in-depth review):

- Ease of use of environment (user interaction, drawing tools, rule construction kit)
- Programming style mixed textual and iconic language
- Semantically enriched graphical rewrite rules
- Analogies for reuse (inherit vs. copy and paste)

The two environments also vary in the ease of use of individual features. Both environments support user interaction through direct manipulation. The programming style incorporated in both environments is mixed textual and iconic language.

After analytic analysis had identified two candidate systems for closer study, an empirical evaluation was performed to obtain more detailed information about these systems. A series of usability evaluations of the AgentSheets and Stagecast environments was conducted using tutorials designed for educators from area school systems. From this study, AgentSheets was chosen as the best candidate for initial study. The empirical analyses were performed in controlled laboratory settings. Participants were instructed to use the “think aloud” protocol during the sessions. Two evaluators observed the experiment in process, collected demographic information from participants and recorded critical incidences from the experiment sessions. One evaluator was in the proximity of the participant in case of serious breakdown. The second evaluator was located in a separate evaluation room. The secondary location was designed so that the evaluator could record critical incidents without distracting the participant, at the same time controlling the audio-visual equipment used to record the session.

4.4. Learning and Reuse in AgentSheets

The third initiative of Phase I, the requirements analysis, was to conduct a study of learning and reuse as a proof of concept for the following: “Will teachers be motivated enough to utilize new educational software?” Also, this study assessed teachers’ modes of learning to program in a direct manipulation environment, the complexity of the user interface for novice programmers, and what techniques and strategies they employed for software and/or component reuse (Rosson & Seals, 2001). This study was utilized to refine the design requirements, and refine tutorial materials for future novice programmer studies. In the study, a small group of public school teachers performed two self-study sessions to learn to use AgentSheets and Visual AgenTalk (VAT). The first session emphasized learning to program with visual direct manipulation programming techniques. The second session introduced the concepts of reuse to create simulations. The reuse trial presented users with two examples, one a concrete presentation and one abstract model. This preliminary study found that the simplified components of the abstract world better supported reuse of programming of simulations by novice programmers.

4.4.1 Introduction to Study

Simulations provide many excellent opportunities for learning, and visual simulation programming encourages even novice programmers to create, hypothesize, and share ideas and understanding of concepts with their instructors. They can be very effective tools to spark creative discourse in classrooms about any subject matter. Recently, a great deal of attention has been devoted to the weaknesses of the U.S. educational system in the areas of science and mathematics. These systems can help address this issue by enabling students to build models to explore scientific and mathematical laws (Resnick, Bruckman & Martin, 1998; Wirfs-Brock, 1995).

Simulation as a learning technology aids teachers greatly in the classroom if integrated into their pedagogy, but this is not a trivial matter. To ease the transition to

using simulation technology in the classroom, both teachers and students will need to be trained in the use of this technology with the aid of a research team (Rader, Brand & Lewis, 1997). In many cases, even when software developers create shrink-wrap solutions that they assume will be useful for the classroom, most new educational software is never adopted or applied (Soloway, 1998; Tyack & Cuban, 1995). This study examined a possible bottleneck to teacher adoption of new simulation software, namely the teacher's willingness and ability to learn how to use a modern visual simulation environment (Rosson & Seals, 2001).

This research was part of a larger participatory design project to develop and evaluate computer support for science education (Carroll & Rosson, 2000). An early finding was that teachers have very little time to spend learning new technology or building technology from scratch, but prefer to adapt content to make it more relevant to their lessons. Therefore, instead of using an exhaustive systems approach to learning to build simulations, minimalist self-study instruction was provided to teachers both to reduce the time they spent in learning to use the environment (Carroll, 1990) and to explore the benefits of reuse programming in a visual simulation programming environment (Perrone & Repenning, 1998).

4.4.2 AgentSheets & Visual AgenTalk

This study investigated the learning and reuse of Visual AgenTalk (VAT), the visual language provided by the AgentSheets programming environment (Repenning, 1994; Repenning & Ambach, 1996; and Repenning & Sumner, 1995). Users create in this environment by developing a set of agents utilizing a simple bitmap based depiction editor. Once created, these agents are the objects that populate a simulation microworld. Since having only generic agents in a microworld is uninteresting, to give the model or illustration life the agents in the world must interact. When creating rules, users utilized direct manipulation techniques to create a set of production rules that guide agent behavior. Each rule in VAT is a production rule that will fire if the precondition is true (e.g., if an agent "sees" a specified agent in a particular position, then carry out the rule). Previous longitudinal research programs in VAT prompted the choice of this language,

since it has been applied in many educational settings (Cherry et al., 1999). This system also includes a compiler that converts VAT simulations into Java applets for convenient web sharing of simulations (Repenning & Ambach, 1997; Repenning et al., 1998).

4.4.3 Example-Based Learning & Reuse

Self-paced minimalist instructional manuals were created to help teachers quickly begin activities and to engage them in meaningful tasks (Carroll, 1990). In creating these minimalist documents, the textual explanation was kept to a minimum to force the learners to make inferences in a guided inquiry session. Using this method both incorporated support for error recovery and exploited the user's prior knowledge (Meij & Carroll, 1996). The examples used in these sessions were realistic models of concepts that would be presented in the normal course of a semester that reinforced skills to be learned. The realistic interaction of agents is relatively complex. Users' tasks were to analyze the agents' behavior and modify them to learn how to make updates to the model. The teachers' final tasks involved using their learning experience to create a simulation of a related problem.

Learning Materials

The learning tutorial was 12 pages long, including a one-page overview of AgentSheets that reassured learners that programming skills were not required and a one-page interaction guide. The interaction guide provided a quick reference, explained the meaning of the icons in the different windows and how to create and test rules, and summarized the meaning of the most common condition and action components.

The tutorial was based on a water cycle simulation (Figure 4.5), which included *sky* as an inert background agent, along with ten other agents. *White clouds* moved across the sky, and absorbed *water vapor*. Water vapor was produced by *lakes* or *puddles*, and when water vapors came in contact with white clouds their moisture level increased. When the moisture level of a white cloud reached a certain level, it transformed into a *dark cloud*, which created *raindrops*, lost moisture, and turned back into a white cloud. *Grass* lost moisture when hit by a sunray, but gained moisture when hit by raindrops. If it

became too dry, grass turned itself into *desert* (and vice versa). The transformation from lake to puddle and back was also a function of moisture content (Rosson & Seals 2001).

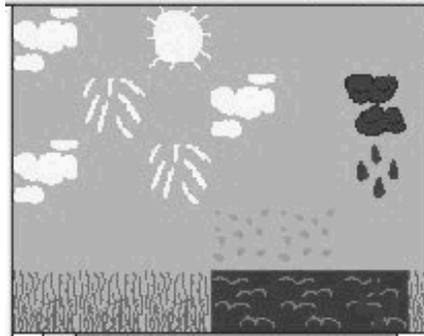


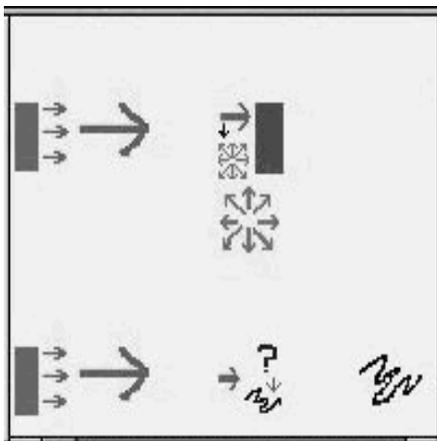
FIGURE 4.5. THE WATER CYCLE MODEL USED FOR LEARNING

Teachers began their activity by simply running the simulation, observing the model's operation, and then exploring the behaviors of agents. Their goals were initially to create a new agent and specify this agent's behavior. Once they had created new agents in the tutorial example simulation, they were given a performance test that required them to create their own simulation from scratch (in this case, a volcano model). Adhering to the minimalist style, there were no explicit instructions on how to implement a volcano model. This was left as a creative self-directed guided exploration for the user, so the experience was unique for each user, building upon their prior knowledge with the aim of reinforcing visual programming concepts.

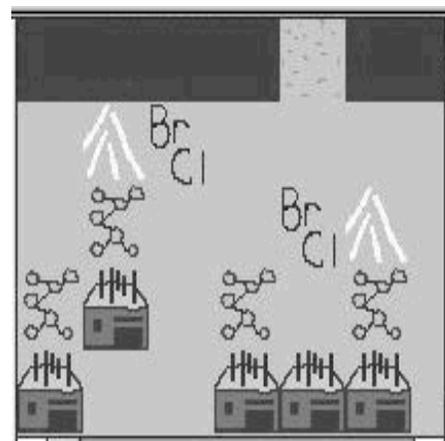
Reuse Materials

The reuse tutorial was 10 pages long with a one-page interaction guide. Users were given refresher material and a reintroduction to the water cycle as a part of their reuse training. The reuse training was to re-explore the water cycle model with a new objective. The aim for this exercise was to introduce them to the notion of recognizing analogies of action between what was represented in the examples and what models they were to create or modify.

The participants began by creating a new agent based upon some action of an existing agent. Each learner then investigated the cloud transformation rules. The objective of this reuse task was to create an animal that worked under these same principles. For example, when the *white cloud* received enough moisture, it transformed into a *dark cloud*; by analogy when a *lion* ate enough *rabbits*, the *lion* grew stronger or bigger. All of the users were able to understand and complete this first reuse task successfully. They were then introduced to the two new models (see Figure 4.6), the generic starter world model and the concrete ozone world model. With the aid of reuse, teachers then created an ocean model and a photosynthesis model.



ABSTRACT STARTER WORLD EXAMPLE.



CONCRETE OZONE WORLD EXAMPLE.

FIGURE 4.6. REUSE MODELS

These two reuse examples were developed to introduce the same action representations to explore the relative comprehensibility and usefulness of concrete, contextualized examples versus abstract examples. The concrete model has the advantage of being connected to the real world by illustrating familiar phenomena. The abstract model has removed the “real world” context and the agents’ names are based upon the behavior they exemplify. The developers of AgentSheets have argued that users will not want to work with abstract agents (Repenning et al., 1998), but preliminary studies of example-based reuse in Smalltalk indicated that the concrete details of realistic usage examples often get in the way (Rosson & Carroll, 1996b).

Participants

Participants in the study were five middle and high school teachers (henceforth P1-P5). They were all teachers in the areas of mathematics and science; one had limited teaching experience. All were familiar with computers, and one had taken one course in FORTRAN many years prior. Teachers expressed positive expectations about visual simulation as a teaching aid (see Rosson & Seals, 2001 for more details).

Procedures

Prior to the experiment, each teacher completed a background questionnaire asking about teaching and computer experience and their expectations concerning visual simulations. Participants were then seated at Macintosh computers in an observation room where the tutorial and Interaction Guide were available. One experimenter sat several feet behind the learners, while another controlled the video equipment and observed through a one-way mirror.

These experiments used the “think aloud” protocol. Participants vocalized what they were thinking, their goals and plans, and their reactions to experiment events. An experimenter in the room took notes. Users were instructed to work on their own. Experimenters would only intervene at times of serious disconnect (i.e., user confused) or system error. The help protocol for handling these disconnects was to direct users back to the materials in general, point to a specific element of the tutorial or screen as a helpful hint, and finally simply to tell the user what action to take.

The experimental data included screen captures from the Macintosh, videos of teachers working, users’ think-aloud comments, and users’ behavior as they worked. Users ended each session by completing a questionnaire describing what they found particularly difficult or easy to accomplish in VAT. In the final set, they were also questioned about both their reuse of material and what sorts of material they would like to reuse in VAT work.

4.4.4 Learning Observations

All the teachers successfully made changes to the water cycle model and created a volcano model. The time spent per user was 55-88 minutes for the first session (Table 4.7). In the user created volcano model shown in Figure 4.7, the basic action is lava moving along a lava tube and, if it sees the up tube, moving into it and erupting.

Table 4.7.

Summary of Learning Sessions

ID	Time (min)	Volcano Agents Developed
P1	88	mountain, eruption, volcano
P2	75	mountain (inert), pressure, smoke, lava, sparks
P3	65	lava, lava-tube, uptube, fresh-rock (inert), smoke, volcano, tree (inert)
P4	80	Volcano, lava (2), smoke, rising-heat, ground-level, plates-under-ground
P5	55	mountain, sparks, pressure, lava

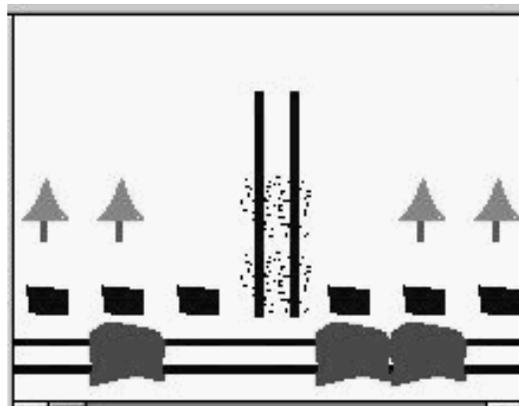


FIGURE 4.7. THE VOLCANO MODEL PRODUCED BY P3

Teachers had some learning problems not attributable to visual programming but, rather, to certain characteristics of VAT (Rader, Brand & Lewis, 1997; Rader et al., 1998). Many had trouble with the direction parameter used in many conditions and actions (e.g., see the rule snippet in Figure 4.8), needing to apply the rule update before testing a new rule, and leaving the simulation running while editing it. Figure 4.9 is an example illustrating the fact that exact location and position were very important in this

visual program. The cloud reacted to water vapor, so water vapor had to be in a specific location (i.e., *below* the cloud). Teachers were also frustrated when, upon logically solving a problem, a system restriction or limitation caused their solution to fail or not work as intended.

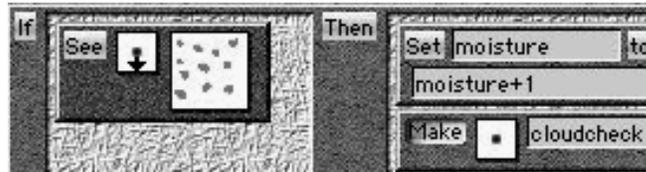


FIGURE 4.8. SAMPLE RULE WITH DIRECTIONAL INDICATORS

A particularly vexing problem concerned the use of the “change” action, which can be used to alter the appearance of an agent. All of our users were attracted to this action, but for the wrong reason—they believed that they could use it to transform one agent into another. The visual feedback a user receives is consistent with this expectation; the original agent takes on the appearance of the second agent. However, the change is in appearance only; the agent’s identity remains the same and it has the same behaviors. Making the full transformation requires a combination action: first the original agent is erased, and then the replacement is created at the same grid position (Rosson & Seals 2001).

Other general conceptual problems were observed. Most participants were able to create discrete visual representation for general objects (e.g., volcano, earth, lava, etc.), but many had a harder time trying to create agents to depict unseen elements (e.g., wind, gravity, heat, pressure). During the volcano project, User P5 commented that “agents are objects that you can see”, but then noted “I’m not quite sure how to draw pressure, so I will make a symbol” (she sketched out a letter P) (Rosson & Seals, 2001). The hope was that a visual environment would encourage the modeling of invisible elements by providing a special “forces” layer of the worksheet, or force-annotation variables attached to visible agents. “Another problem stemming from the visual character of the simulation language concerns agent composition. In the volcano project, several teachers used a transparent overlay metaphor to model the volcanic eruption—a mountain was drawn, and then pieces of the explosion image were carefully drawn so as to fit “on top”, with

lava down the side, and so on. VAT supports this overlay strategy; the unused space in a cell is transparent by default; agents can be superimposed to create a composite image. Participant P2 was particularly adept at creating such a visual overlay, carefully drawing agent depictions so that they would line up correctly (Figure 4.9). However, even though the visual display may look right, the rule engine only computes behavior for agents in the “top” layer” (Rosson & Seals, 2001).



Figure 4.9. A volcano built in visual layers

VAT is an object-oriented language that utilizes agents that operate independently in a simulation micro world or worksheet. Each agent within this environment has its own independent behavior. Many teachers had problems grasping that each agent was responsible for its own behavior. They had particular problems in dealing with cooperative behavior between agents. For two agents, such as an ocean that creates waves, participants must realize they are cooperating to produce the desired effect. The ocean creates waves, and then these waves move forward, but when the wave hits the beach, it is the beach that converts the wave into a ripple, or should the wave transform itself? A related problem is the distributed thread of control typical of object-oriented systems (Rosson & Seals 2001).

Many users had to create a similar analogy when the volcano created lava, and the lava, upon contacting the ground, could enact another transformation. With the volcano problem, the teachers generally had a sequential plan. For example, P2 declared, “first the lava moves to the right position, and then the smoke goes up...” (Rosson & Seals, 2001). However, there was no place where she could write out that plan; instead she had to work agent by agent, in a test and debug fashion to figure out how to get one agent to trigger the appropriate next step. The problem of designing and understanding distributed control structures is well-known in object-oriented design (Wirfs-Brock, 1995) and has led to the

development of sequential design representations such as use cases and interaction diagrams.

In the final user reactions to AgentSheets, all the teachers were positive about using this environment as technology to enhance their curricula. They also gave implementation details of a basic simulation. Most indicated, however, that they would need more exposure and tutorial time to the tool to feel comfortable enough to utilize it within their classrooms.

4.4.5 Reuse Observations

In their second sessions, most teachers worked more smoothly with VAT, having mastered many of the small details (e.g., creating new agents, adding rules) in the learning sessions. The duration of the experiment tutorial was 45-90 minutes, and all participants created both the food chain simulation and the ocean world, with considerable degrees of variability in the sophistication of created simulations (Table 4.8).

Table 4.8.

Summary of Reuse Sessions

ID	Time (min)	New Agents Developed
P1	90	<i>Food chain:</i> wolf, rabbit, big-foot (inert) <i>Ocean world:</i> ocean, waves, beachsand, ripples (inert)
P3	65	<i>Food chain:</i> lion, rabbit, sleeping lion (inert) <i>Ocean world:</i> ocean, waves, beachsand (inert), ripples, water (inert)
P4	60	<i>Food chain:</i> lion, rabbit (inert), big-lion (inert) <i>Ocean world:</i> ocean, waves, beachsand, ripples (inert)
P5	45	<i>Food chain:</i> lake (borrowed), lion, rabbit, big-lion (inert) <i>Ocean world:</i> ocean, waves, beachsand, ripples, water (inert)

The teachers' first reuse task was to build a food chain using the water cycle as a model, and they were encouraged to study the cloud's behavior and reuse it as a model for predator growth. All teachers were introduced to this concept of reuse. Some tried to use this form of reuse by example, but many solved the problem using different approaches. All four of the teachers produced working food chain models with a lion that "grew larger upon eating a rabbit," but only P5 made this change contingent on an internal variable. In hindsight, this is not surprising—although the teachers had studied the cloud before, this was the first time they had needed to work with variables.

Even though users utilized an alternative technique than that expected, many users had an immediate grasp of intended concepts:

- P1 grasped the conceptual similarities of water cycles and food chains, "both have cycles, both have products going up the chain, decomposition and so on," but decided that reuse was too complex and stated, "OK, I'm just going to work from scratch."
- P4 did not reuse the cloud example, but copied bits and pieces of the cloud's rules, in a sort of trial-and-error process.
- P3 and P5 quickly grasped the rules for tracking moisture content and were able to correlate it as analogous to a lion's "fullness."
- P5 was very successful at adopting a systematic approach, copying the relevant rules and adapting them to fit the food chain context.

The reuse efforts observed during the food chain project illustrate five general strategies for reuse (Table 4.9). Two of the teachers described a high-level domain analogy—that these two "cycles" are similar. One teacher voiced a purely visual analogy, wanting to reuse the depiction for an existing agent for a new one (VAT supports this). Three of the four participants exhibited direct agent reuse, bringing the water cycle's sky, grass, sun, clouds, and so on, into the background of their food chain. This proved interesting because it introduced water cycle functionality into the food chain, even though it was not called for by the food chain problem (i.e., the clouds produced rain, the grass became desert). One teacher worked in a highly opportunistic fashion, a strategy termed in this study an ad hoc analogy, and browsed the rules of existing agents to

generate ideas for conditions or actions she could try out. This was generally unsuccessful, but did produce a benefit later on when she recalled something she had seen working earlier. Two teachers worked with the suggested behavior analogy, attempting to adapt the cloud-absorbing-rain mechanism to the lion-eating-rabbits case.

The food chain project included explicit prompts for reuse: use the white cloud as a model for a lion eating rabbits. However, for the second reuse task (ocean world), no prompts were given other than the general advice to reuse material (from the ozone or starter worlds) if useful. The teachers' responses to this advice are summarized in Table 4.10, and reflect considerable variation. As the table suggests, the two teachers working with the starter world showed considerably more reuse than the two using ozone world. P1 did not even want to consider reuse until prompted. P4 did look carefully at the ozone agents, but did not return to them during her work. In contrast, both P3 and P5 used appropriate starter world agents as models for ocean world components.

Table 4.9.

Reuse Strategies Observed During the Food Chain Project

Reuse Strategy	Description
Domain analogy	The problem domain is recognized to have a structural similarity to a problem domain for which there is already a solution, concepts are mapped from domain to domain
Visual analogy	A component is recognized as having the same appearance but different behavior as an existing component, so the depiction is reused to save editing time
Agent reuse	An entire agent is co-opted into a new problem as-is, even to the extent of accepting behavior from the old domain not needed in the new problem
Ad hoc analogy	Bits and pieces of rules (individual conditions or actions) are seen in working agents and tried out in the new problem to see if they might do something useful
Behavior analogy	A behavioral mechanism for one agent is used as a model for an analogous mechanism, either as a plan (the user follows the steps but creates her own code), or more directly by copying and editing the rules, to adapt them to the new context

It is tempting to conclude from this that the starter world was more “reusable” than the ozone world. Recall that these models contained the same functions, but ozone illustrated this function within a realistic context. In her questionnaire, one starter-world user emphasized how she liked “seeing a model that was abstract—it opens possibilities for really abstract models”. The other said “I feel that I learned how to transfer behaviors to new modeling situations to build on previous models.” In contrast, the teachers using ozone world remarked “I copied some of the patterns, colors” and “I found it easier to create new ones.”

Table 4.10.

Reuse During Ocean World Project

User	Individual Approaches to Example Reuse
P1: Ozone	<ul style="list-style-type: none"> - examined Ozone agents only after request made - developed ocean agents entirely from scratch
P3: Starter	<ul style="list-style-type: none"> - carefully examined Starter agents - modeled wave on mover, first modeled ocean on changer, but later decided emitter was a better model
P4: Ozone	<ul style="list-style-type: none"> - carefully examined Ozone agents - developed Ocean World from scratch, “I found it easier to create new ones-a bit less confusing” - never examined Starter agents, able to infer behavior correctly just from names and testing
P5: Starter	<ul style="list-style-type: none"> - voiced relations, “an ocean that makes waves would be similar...an ocean might be the emitter” - explicitly modeled ocean on emitter, wave on mover, and beach sand on transformer

However, these findings must be qualified by noting the small number of teachers studied. The teachers who used the starter world (which utilized an alternating assignment heuristic) happened to be the two who seemed to best appreciate the concept of reuse because they seemed to recognize and benefit from the analogy between the cloud and the lion in the food chain. Thus, these findings are intriguing, but clearly require further study.

4.4.6 Discussion

Teachers were able to learn VAT, and use this system to create simulations from self-study materials in 55-90 minutes. They analyzed and modified a sample simulation, and created their own simulation from scratch. In a second session experiment, they built two new simulations reusing example simulations. In general, all users were positive about their experience and could imagine a role for VAT and AgentSheets in their own classes. Findings indicated that teachers could incorporate this type of visual simulations into their pedagogy.

Teachers as Simulation Builders

This study pictured teachers as simulation builders who, as content matter experts in their classrooms, were the best candidates for designing educational simulations. Research here worked to reduce teacher frustration with programming. In many instances, teachers knew exactly what they wanted to model, and could create this functionality in the current AgentSheets computational model. Many teachers' domain analysis of the problem was very robust and sophisticated, but most rendered very simplified version of their models. However, since students' knowledge of the domain was limited, they would build structures that simply "look right" (Lewis et al., 1997).

With current tools, teachers might have difficulty transferring their elaborate mental models into working simulations without greater programming skill. In order to mitigate this difficulty for teachers, improved instruction with an example-based tutorial presenting different simulations of the same phenomenon could be provided that included variations in complexity and coverage. This would help teachers understand that a range of models can be generated from simplified problem analyses. The types of mechanisms, which may be easy or difficult to model, could also be explained; however, this technique would limit exploration, which conflicts with a minimalist hands-on approach to learning.

Even though these are simplified approaches to programming, many of the problems the teachers experienced appear to be similar to issues discussed for years in the

literature on OOD and programming (Rosson & Carroll, 1996a; Rosson, Carroll & Bellamy, 1990).

Analyzing and assigning responsibilities, determining how the agents should collaborate, developing and implementing an overall thread of control—all of these are general problems when working with networks of interacting objects. It may be that environments like VAT can benefit from the work on design representations and tools developed for professional programmers using more sophisticated and general object-oriented languages and environments (Rosson & Seals, 2001, p. 243).

Example-Based Learning of VAT

All of the users were able to understand VAT basics (i.e., concepts of agents and rules, changing rule parameters, etc.). Most were able to understand basic rules, but many had problems understanding compound rules and rules where multiple agents work cooperatively. This was frustrating to some novice programmers; one teacher complained “this is like a math equation!” (Rosson & Seals, 2001, p. 243).

There is always an inherent tradeoff in providing examples that are simple enough for novices to understand, yet complex enough to be realistic (Carroll & Rosson, 1991). In this case, realism was chosen, but the presentation perhaps should have included more scaffolding to support novice user understanding. The following section illustrates how the *Cloud* agent was introduced. A comprehensive description of the cloud’s behavior was presented together, although it could easily have been broken into sections. In particular, an analysis of the moisture-tracking mechanism could have been offered *at the point* that users were attempting to use it as a model in their food chain projects. In some cases it was simplified at the expense of realism: the cloud agent in the water cycle keeps track of its moisture; when moisture gets to a certain level, it transforms itself into a *new* agent (a black cloud) with new behavior. A more realistic model might be a cloud agent that just changes its appearance as its moisture increases. However, such behavior would be more complex, because it would have more rules and checks, and everything it does

would be contingent on an internal variable. The two different sets of behavior (for the cloud as well as for other agents such as grass and water) were modularized in order to make the example code easier to understand. As a side benefit, this departure from the real world situation made it possible to illustrate the general strategy of agent “transformation (Rosson & Seals 2001).

The results indicated that many issues of example-based learning can be addressed by modifying the environment. Teachers had a great deal of difficulty in managing the many small windows in VAT, with the most recurring problem being the tendency to confuse the programming resources (the condition and action palettes) and whether these resources were attributes of the example project model or the project being created. Similar problems existed in the traditional Smalltalk development environment. Programming experts often developed elaborate housekeeping strategies, using screen space to organize multiple related activities. In studies on simulation reuse, there were attempts to ease this problem by creating a special documentation-oriented browser onto resources that was visually distinct from the editing area (Carroll & Rosson, 1998).

Reuse in Visual Simulation Programming

Findings on teachers’ willingness and ability to reuse example code in their own work were quite promising. Only one of the teachers had problems with understanding the reuse applications. This supports the theory that in general, reuse would improve the usability of the environment for many teachers. In the two reuse trials, the abstract starter world was easier for users to decompose. They could easily deduce the behavior of the agents from their depiction and action, versus having to study the rules in detail with the example based reuse sample.

The broader implications of the reuse study suggest that end-users may benefit from a “pool of aptly-named abstract agents” (Rosson & Seals, 2001). With this pool of reusable agents, the teacher would be able to browse and select reuse agents that best fit the models for their specialized curricula needs, reminiscent of the prototype paradigm for object-oriented programming (e.g., Self (Ungar & Smith, 1987)). It appears that end-users such as teachers have the ability to create simulation models, but are unlikely to

create many agents or simulations on their own because of time and motivational constraints. By providing a reusable framework and simulation library, teachers will be assisted in adopting visual simulation as a useful educational technology in the classroom.

4.4.7 High Level Summary of Usability Findings

The study of AgentSheets yielded many valuable insights in addition to the aspects of usability presented. On average, most of the simulations were simple, with only four of the participants taking the time to produce an interesting working model. There were several critical incidents that indicated the need for a redesign to improve teacher performance. These incidents are broken down into the three areas of Drawing and Icons, Rules and Methods, and General Environment.

Drawing and Icons

Drawing and Icon critical incidents were problems noted during object creation and drawing of objects in the microworld. Most participants had no problem drawing after they realized that ink color was not an indicator of color, but a color palette. This problem was caused by icons being too small. For example, the apply step button and trash can icons were smaller than the standard 32x32 pixel representation used by desktop icons. There was a similar problem with the bucket fill tool, where the fill point was the tip of the paint, rather than the entire bucket.

Most were frustrated by having to use a single pixel drawing tool and a single pixel eraser. A variable size drawing tool would have been more desirable and helpful. Most users did not recognize the thumbnail of their drawing. There was also need for support of the undo and reset command in drawing mode. Beginning a drawing in this environment was a problem for all due to the multiple windows involved. When selecting something, the drawing window had first to be brought into focus, then the object selected, and then the worksheet clicked on twice. The first click brought the worksheet into focus, and the second click allowed users to begin to draw. All the users became confused as they tried to drag objects from the gallery to the worksheet, as when rules are

created. Most of the time they dragged the object out of view by accident in their attempt to draw the object onto the worksheet.

The next problem was sizing of agents. In the AgentSheets environment, users are restricted to only a single agent size within a simulation (i.e. a 32 x 32 pixel representation). In Volcano construction, most users wanted this object to be larger than the rest. A reset function would have helped instead of having to create a backup worksheet to reopen. Users also tended to leave the simulation running in the background while going on to other tasks, which slowed the response time of the system.

Rules and Methods

Rules and methods critical incidents were problems noted during rule creation or testing behaviors of agents in the microworld. Rule ordering and use of rules was a hard concept for them to understand. When creating rules, users had a problem with having to make rules that checked the topmost agent only. The majority of user rules would have worked if the environment recognized all objects in the same *xy* location when performing the check. The rule to simulate non-determinism (i.e. percentage chance) was not understood by most users until explained. Only one user used more than one method. Most users used the direction finder properly the first try. However, once again, they had a problem with Undo. Users would often like to be able to delete objects and start over, but with no undo and only pixel erase users tended to avoid the problem by creating an alternative agent with a similar name. When making changes to an object, if the depiction editor is closed it does not redraw all objects. They only redraw if required or forced. Users must either redraw individual agents or instruct the windows to redraw themselves.

General Environment

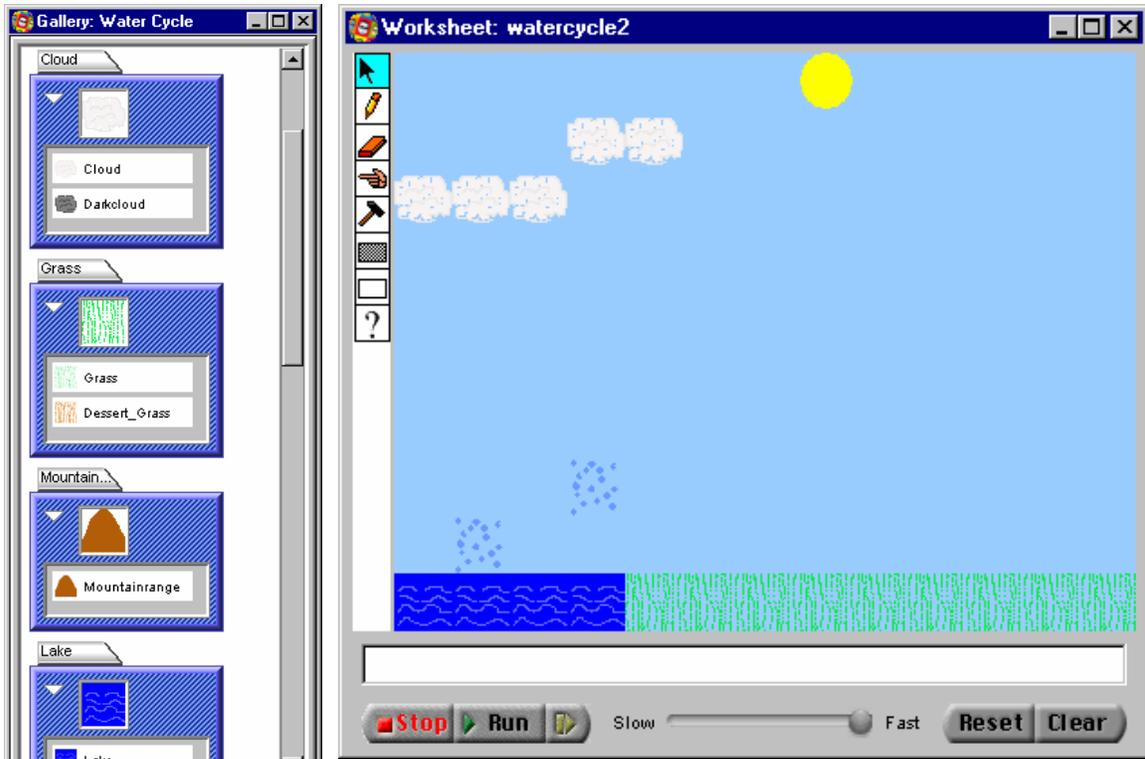
The biggest problem, which caused many users to become disorientated, was having to use too many windows. AgentSheets' main interface is a simulation microworld where objects interact, and there are six windows that users manipulate as they work with the environment (see Figure 4.10). Users program using direct manipulation, dragging conditions and action templates to a rule window. At times, they may need to use the window menu to navigate the many windows on screen. Users also

have problems selecting agents because windows must first be brought into focus, then agents can be selected. This often caused users to select the wrong agent or, in one case, no agent at all. An error dialog box pops up when users drag rules across the depiction editor.

Rules are highlighted in yellow and black when they are being used. This is intended to denote caution, but it may be preferable to highlight the rules in green, as this coloring caused at least one user to be hesitant. Most users had to read the manual to find that the arrow tool was used to move objects, as this manipulation technique was not intuitive. One user complained that he did not like the fact that the pencil tool was used to create the object and then used like a copy-paste to draw it on the worksheet. This icon overloading, where more than one interaction is mapped to the same icon, is confusing and in one instance the user's final work was lost. The system should prompt users to save their object gallery. The "transparent" function did not work as expected, and the reset function, which did not work properly, was only available when the simulations were saved as an applet. Dragging rules proved a problem if users did not grab rules in the center but on a part of the rule that could be manipulated.

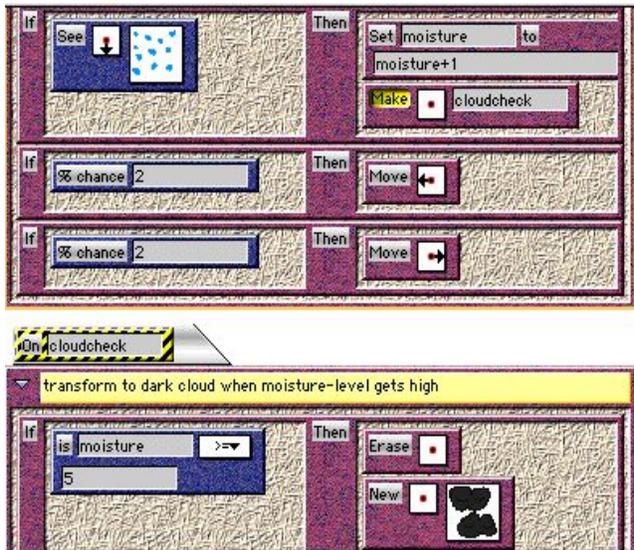
4.4.8. Implications for New Simulation Environments

The artifacts of the learning and reuse study of AgentSheets were used to generate the requirements for a new visual programming environment that will better support this novice community and provide a framework for reuse. The participants were trained with minimalist instructional materials that functioned as a basis for developing simulation materials for the new environment. The aim was to enable teachers to create real simulations quickly, relying on their domain expertise and the new skills they have learned about simulation creation. The rationale for building simulations as educational material is practical. Kuyper (1998) found that simulations are independent of time and place, which makes them more readily available for real experience. He also considers that simulations can provide a better conceptual model of a situation, and can be used to create virtual environments.

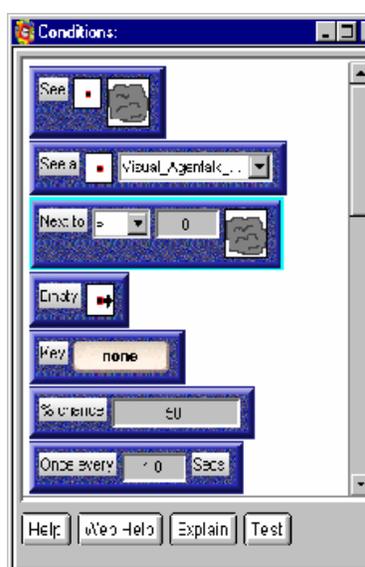


Worksheet

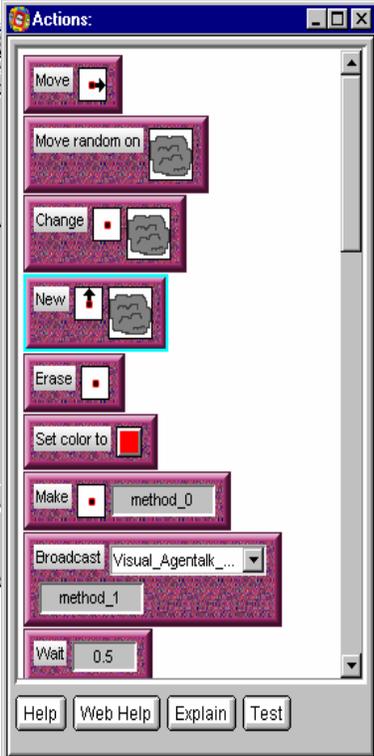
Gallery



Rule Window



Conditions



Actions

FIGURE 4.10. AGENTSHEETS INTERFACE

In this research, the following general problems with visual languages were identified: the environment, drawing tools, and rule creation (see Appendix B for details). A problem was also encountered with the level of abstraction. Most of the environments studied were based on a grid-based concept and objects that when layered did not operate as anticipated (e.g., rain on top of a flower in one case would affect only objects in the top layer of the representation). Minimalist instruction was used to creating tutorial materials for empirical analysis (see Appendix G, Appendix H for tutorials). The tutorials created were very helpful to users and got them started quickly, aiding them in completing the exercises with salient results. Ideas were also explored for the building of reuse libraries by providing categories for types of problems. Since this new environment was created for secondary school teachers, simulations were categorized based on content areas, such as Mathematics, English, Biology, or Social Science. The first category to be built into the library was Physical Science. Within each category, the plan is for simulations to be organized in alphabetical order. The identification of general problems with visual languages in this study helped to build our knowledge of visual languages.

4.5 Learning Study of Stagecast Creator

The fourth initiative of Phase I, the requirements analysis, was to study programming by demonstration and direct manipulation techniques of the teachers' pupils and community seniors. This was designed to build our knowledge of novice programmer usability problems that can be encountered in any graphical rewrite rule visual programming environment. Even though the students were not the primary subjects of the research and design efforts, they will also be users of the final system. As with the previous research for this study, the knowledge gained will be utilized to refine the design requirements and tutorial materials for future novice programmer studies.

4.5.1 Introduction to the Stagecast Creator Study

The learning study of Stagecast Creator was designed to create a cross-generational learning community working together to design, construct, and discuss

simulations of community interest (Seals & Rosson, 2002). This effort included a requirements analysis of the minimalist training materials, in which groups of students worked in groups on two related tutorial modules. The aim was to identify ways to motivate these novice programmers to create simulations, and continue to make their own contributions to the virtual learning community. The students were generally successful in their work with Stagecast and reported that they enjoyed their experience. It seems likely that if students have fun building simulations they will spend more time in the environment and with continued experience learn more about visual simulation programming.

4.5.2 Stagecast Creator

Stagecast Creator is based on a movie metaphor and users create a cast of characters who interact within a simulation microworld. Users create Stagecast simulations with a macro-recorder device that allows users to program by demonstrating and example, and also by direct manipulation. To demonstrate a rule, the user selects the character to be programmed and a bounding box or “spotlight” highlights the character to be edited (Figure 4.11). For example, if the user wants a cloud to float across the sky they would simply drag the cloud forward which would record a new rule for this character. This makes it very easy for novice users to begin rule creation, although there are still some semantic complexities that arise. The spatial context and visual appearance are requirements of the rule’s precondition. For example, if two characters are next to each other while a rule is being demonstrated, both objects would always have to satisfy the precondition or the rule would not be executed.

4.5.3 Minimalist Instruction for Stagecast Creator

The instructional method for this group was a minimalist tutorial, which emphasizes that people learn well by engaging in real activities. Smoking was a big problem in the middle school concerned, so the tutorial materials were based upon this subject, using this example and supporting material to instruct students in learning to use this environment. In creating the minimalist tutorial as an active learning assignment, the instructions included getting the user started quickly with an action oriented task,

minimal verbiage, support for error recovery, and open ended questions as opportunities for exploration. Details of procedures were only given once, so that the user would have to test their active recall.



Figure 4.11. Stagecast Creator Rule Creation

4.5.4 Procedure

The participants of the Stagecast Creator study were ten middle school students who performed their activity in pairs, with a navigator and driver of the experience. They were instructed to use the think aloud protocol so that the facilitators would receive insights about their cognitive processes and their opinions about their experience, and be able to track their location in the tutorial and understanding of the documentation. The study was performed in a usability testing room and the facilitators were in a separate evaluation studio, where the experiment was recorded by video and audio. Critical incidents were captured during the experiment to reveal pros and cons with the software and/or tutorial, to validate the design and usability, or to suggest areas for improvement or redesign. After they completed the minimalist tutorial as a guided exploration through their experience, they were given a survey as a high level assessment of their learning experience to gauge what they learned, and reveal if there was any indication of direct transfer to the learning of basic programming skills.

4.5.5 Observation and Results

This preliminary study of Stagecast identified some of the general problems of visual programming. Some of the problems experienced by students highlight general characteristics of visual simulation programming that may be problematic for novice programmers. One of these is the inherent relation between internal variables and the visual state of the simulation. In Stagecast, global and character-specific variables add powerful functionality for tracking and modifying a simulation's state. As a result, the sample simulation developed emphasized the use of variables.

Some performance tasks set by the tutorial were devised to ascertain if students understood the logical relation between a variable and the visual state apparent on the screen. For example, in one case, the tutorial indicated that students investigate a rule and explain how "sickness" is tested. In this case, there was a test that determined whether the variable was greater than or equal to 10. Some students did not understand this concept; however, one student pair did grasp the relation between the variable and its value with the character's appearance, which, in fact, changes as a consequence of changes in this variable. One learner stated, "The sickness is tested by the ...," and her partner continued "... appearance of the character."

Another problem that stems from the visual character of Stagecast programming concerns the role of visual context in the GRR or before-after rules. As emphasized earlier (Figure 4.11), characters must be in exactly the right position and exhibiting the correct appearance in order for a rule to fire. Often, students did not seem to understand how important it was to get this context exactly right. For instance, when asked to create a cloud and make it move, one student positioned it on top of another character (in this case, the basketball hoop) because he liked the way it looked. As a result, a rule demonstrated for the cloud then included the basketball hoop as part of its visual context, making the rule too specific to be useful.

Participants spent between 30-55 minutes working through Activity I and 34-47 minutes in Activity II. All participants had previous computer and drawing experience. In

general, most of the students were quite successful and engaged with the Stagecast environment and with the tutorial tasks. However, this general success emerged in spite of various problems, some specific to individuals, others more general. The most interesting problems and successes observed are discussed below.

In the Knowledge Survey and exit interview, students expressed that Stagecast was easy, fun to use, and that they would like to use it in their classes; but they also indicated that they needed more exposure to feel confident of their understanding of the environment. Thus, while these training materials are clearly on the right track, more help needs to be provided.

The study developed a minimalist tutorial for the Stagecast Creator environment that first introduced learners to visual programming through study and modification of a well-designed example simulation. Subsequent activities guided learners through creating a new simulation on their own. All of the learners, in this case middle school students working in pairs, were quickly engaged by the Stagecast activities. Most began to explore on their own within the first few minutes of their tutorial. At the same time, they were able to refer to the tutorial to find new suggestions for activities, which kept them moving forward and learning new skills. Thus, the tutorial was successful in encouraging active and exploratory learning.

The minimalist strategy of anticipating likely errors and inserting recovery information also seemed to work. While students got "off track" often, they almost always were able to either recover on their own through further exploration or through use of the tips provided in the tutorial (e.g., re-setting the simulation became a common "fix"). There were only two instances in which experimenters needed to provide assistance. One instance was a singleton learner who was unable to answer one of the questions posed by the tutorial and began repeatedly looping through a set of actions. The experimenter advised him to simply continue. The second was a pair of students who had trouble re-ordering a rule's action list. These sorts of specific user errors, often uncovered during minimalist learning studies, will be addressed by the next iteration of the tutorial.

In this study, although the details of the Stagecast Creator learning tasks were only given once, users seemed to have no trouble completing tasks a second time on their own. An attempt to minimize text by inserting icons and screen shots seemed to work well for these users. They often skipped the text to find the next visual instruction to mimic. This enabled rapid exploration, but at times caused problems, such as when users had to come back and re-read to find key explanations (Seals & Rosson, 2002).

4.5.6 Discussion

It is important to find new ways to promote collaborative learning generally when recruiting additional students and other community members into the project. Investigating gender issues may also be interesting, especially since only one girl was interested in participating in this experiment, and the pairing that included the girl made the best use of the minimalist manual by carefully utilizing the help provided. The subjects were self-selected volunteers, and it seems likely that boys are particularly interested in computer programming and computer game type activities.

The students recruited were quite successful with the Stagecast tutorial, but there were a number of issues raised that must be addressed by future work. Some of these were basic issues related to the visual programming paradigm, for example, the relation of internal variables to the visual state of a character, or the role of specific visual context in creating rule preconditions. Other problems were more specific to Stagecast, such as confusion about icons or the management of many small and similar windows. Opportunities for improving the tutorials error detection and recovery information were identified, for instance when guiding the learners in re-ordering actions (Seals & Rosson, 2002).

4.6 Longitudinal Guided Exploration of Stagecast

Efforts to elicit more requirements for the development effort and to become familiar informally with end-users' informal domain of discourse led to a longitudinal guided exploration of Stagecast Creator. The goal was fundamental, to provide a better understanding of users and their particular needs. Most longitudinal studies are conducted in the form of ethnographies, but such an evaluation may take weeks or even months. When user interface designers work with users to elicit design suggestions this effort transforms into participatory design, and designers tend to limit this process to days or weeks. During this guided exploration, teacher developers will be the end-users and beneficiaries of this design effort.

To refine the materials for the system and supporting tutorial materials, content material and simulation building were discussed with two middle school science teachers. Since these development studies focused on teachers in grades six through nine, two teachers who participated in the formal evaluation, and thus were already familiar with visual programming systems, were selected. As with participatory design, it is preferable to obtain requirements for simulations from representatives of the potential user group, and hear their perceptions on what they expect from a software curricular aid and ideas for simulations. During 6 sessions, each ranging from 2 – 4 hours in length; the teachers were introduced to a visual programming environment and given instruction to enable them to bridge the Zone of Proximal development (Vygotsky). There was considerable distance between what the users knew initially and their potential for knowing and creating in the area of visual programming. the aim was to increase their level of competence in this area by coordination with the proficient or a more knowledgeable other (McMahon, 1996).

In order to gain a better understanding of the teacher participants, their comfort level with computers, and novice visual programming of simulations, a longitudinal study was performed with the two middle school science teachers, scheduled for 10 weeks. The

tool that they were to investigate was Stagecast Creator. Their experience began as a typical tutorial with guided exploration cards to guide their self-study. Each teacher worked individually for his or her learning session. Their first learning session used the minimalist tutorial in the form of guided exploration cards, guiding the teachers through starting up the system, opening a micro-world example, interacting with that example and examining the rules, and creating new rules.

For their second and third sessions the teachers explored some of the system and facilitator supplied examples. Once the teachers were more familiar with the environment and some simulation examples that had been created, they were asked to define a set of requirements. Specifically, they were asked what topics they planned to cover during the year and of these topics, which did they think would be good candidates for educational simulations. Their fourth session began by reviewing the teachers' suggested simulation topics (Table 4.11).

To give the teachers a tangible task for their next four sessions, their task was to create two simulations that they would use in their first month of science classes. For both teachers, their first in-class experiment involved scientific inquiry. The teachers illustrate natural phenomena with a small experiment, ask their students to explain their observations, and finally explain the scientific phenomena. The first set of simulations created by our teachers thus dealt with the area of scientific inquiry and their first two classroom experiments were to illustrate "cause and effect" scenarios. Scenario one was "If you add water and sunlight to a plant, then the plant will grow" as an illustration of photosynthesis. The second simulation scenario, "Raisins in ginger ale", was an illustration of scientific inquiry as the students tried to work out what was happening.

Table 4.11.

General Outline of Topics For Middle School Physical Science

<i>First Semester</i>	
I. Scientific Method	<ul style="list-style-type: none"> - observation/inference (simulation with raisins & soda) - experimental design (simulation with flower & sun) - Gas Laws (Boyle's Law, fluid dynamics & Charles Law) - density - solubility
II. Physical Properties	<ul style="list-style-type: none"> - states of matter - bp, mp, gp (time vs temp) - color, odor, texture - viscosity, malleability - adhesion, cohesion (water molecules stick to each other)
III. Atomic Theory/Models	<ul style="list-style-type: none"> - parts of the atom - Dalton, Thomson - Rutherford, Bohr, cloud models - Periodic table
IV. Chemical Properties	<ul style="list-style-type: none"> - rate of reaction (surface area, temp., concentration) - chemical reactions (synthesis, decomposition, single replacement, double replacement)
<i>Second Semester</i>	
I. Motion	<ul style="list-style-type: none"> - speed/acceleration - gravity (show how weight changes with different gravitational forces) - Newton's 3 Laws (inertia, action/reaction, $F=ma$)
II. Waves	<ul style="list-style-type: none"> - wavelength, frequency, velocity - reflection, refraction, diffraction, interference - types of waves (compression, transverse) - types of waves (sound, volume & pitch; light, optics)
III. Electricity	<ul style="list-style-type: none"> - circuits (series, parallel) - conductors vs. insulators - electromagnetism
IV. Simple Motors	<ul style="list-style-type: none"> - types (levers, pulleys, wheel & axle, inclined plane, screw, wedge) - mechanical advantage - efficiency
V. Nuclear Power	<ul style="list-style-type: none"> - fusion - fission (radioactivity, half life) - issues of nuclear safety
VI. Alternative Energy	<ul style="list-style-type: none"> - solar - wind - geothermal

The illustration of photosynthesis was described as follows: You begin the experiment with the sun shining and a flower in a pot. With the sun shining, as the gardener waters the flower it grows. This phenomenon is caused by photosynthesis, when the chlorophyll in the plant receives sunlight and water; this is transformed into energy and causes the plant to grow. In the simulation the gardener walks toward the watering pot, waters the flower, and the flower grows in a three-stage animation. The photosynthesis scientific method experiment is illustrated in Figure 4.12.

The second simulation scenario, “Raisins in ginger ale”, was set up with ginger ale and raisins in a glass. The scenario is that ginger ale is poured into a glass and then the student adds raisins to the glass. The effect of carbonation in the water will cause the raisins to bounce to the top of glass propelled by carbon dioxide bubbles. Once the bubbles reach the surface they will burst and the raisin will fall back to the bottom of the glass. This cycle repeats until the level of carbonation is too low and eventually all the raisins settle at the bottom of the glass. This scientific method experiment is illustrated in Figure 4.13.

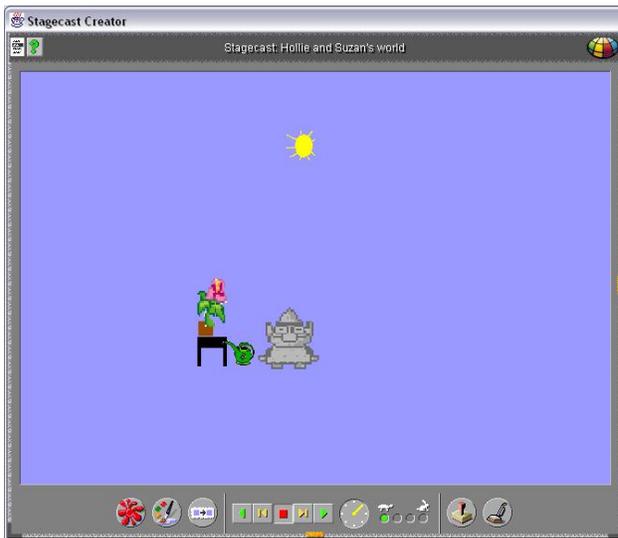
When the teachers began trying to build the simulation, they first asked “Do we have to draw a glass?” As the aim was to investigate their simulation creation techniques, not their drawing ability, they were reassured that if they could find usable images they could use those, but any that were not available would have to be created. They next asked “Where can we get a big glass to put the raisins in?”, so the next task was to go to the web and do a Google search. The teachers found an image that was similar to what they wanted, but much too small. With assistance, they were able to import their desired graphic of a wine glass that was very similar to the glass of ginger ale in their requirement, which they just had to resize.



A.



B.



C.



D.

FIGURE 4.12. TEACHER CREATED PHOTOSYNTHESIS

Next they created a raisin and, as the final object in their simulation, they needed to create bubbles as the concrete artifact of the carbonization effect. In scientific simulations, finding a physical representation of some of the abstract concepts or components, and devising a realistic depiction of it in a simulation is generally the hardest aspect of simulation creation. The teachers created a drawing of a bubble, but wanted the bubble to mimic the activity of carbonation in ginger ale. They wanted to

illustrate that a bubble would burst when it reached the surface and successfully created an animation of the bubble popping. Their final task was to create a rule for the behavior of the simulations. The raisins would begin at the top of the glass filled with carbonation bubbles. They created two rules, one for raisins to begin falling and another for the bubbles to begin to rise. These rules are adequate as long as their objects encounter no obstacles. Then users have to work around a problem that is commonly encountered in direct manipulation visual programming systems, the problem of rule brittleness caused by exact visual mapping. The teachers created rules to give their objects behaviors to perform actions and interact within the problem. With this type of production system, if a certain visual precondition is met, then the rule is fired and the resulting action is carried out. If there is no visual match then the rule will not fire. The problem of rule brittleness arises because the rule will only work in one situation and if there is a subtle change in semantics a new rule will have to be created for the new situation. With the teachers' problem, the raisins fall if there is only ginger ale below them, but when there is a subtle change in the rule, where a visually different depiction of a bubble is below the raisin, the user has to create a new rule for this new visual instance. So instead of being able to respond to a general rule, "if any object is below it, fall", it can only respond to one specific visual depiction of the bubble. With the exact visual mapping requirement, if a different sized bubble is below the raisin, the user would have to create a new rule for this situation. Even though they were frustrated with problems caused by the rule brittleness problem, the teachers were able to create several new rules to match all the new visual situations and successfully ran their carbonization experiment simulation.

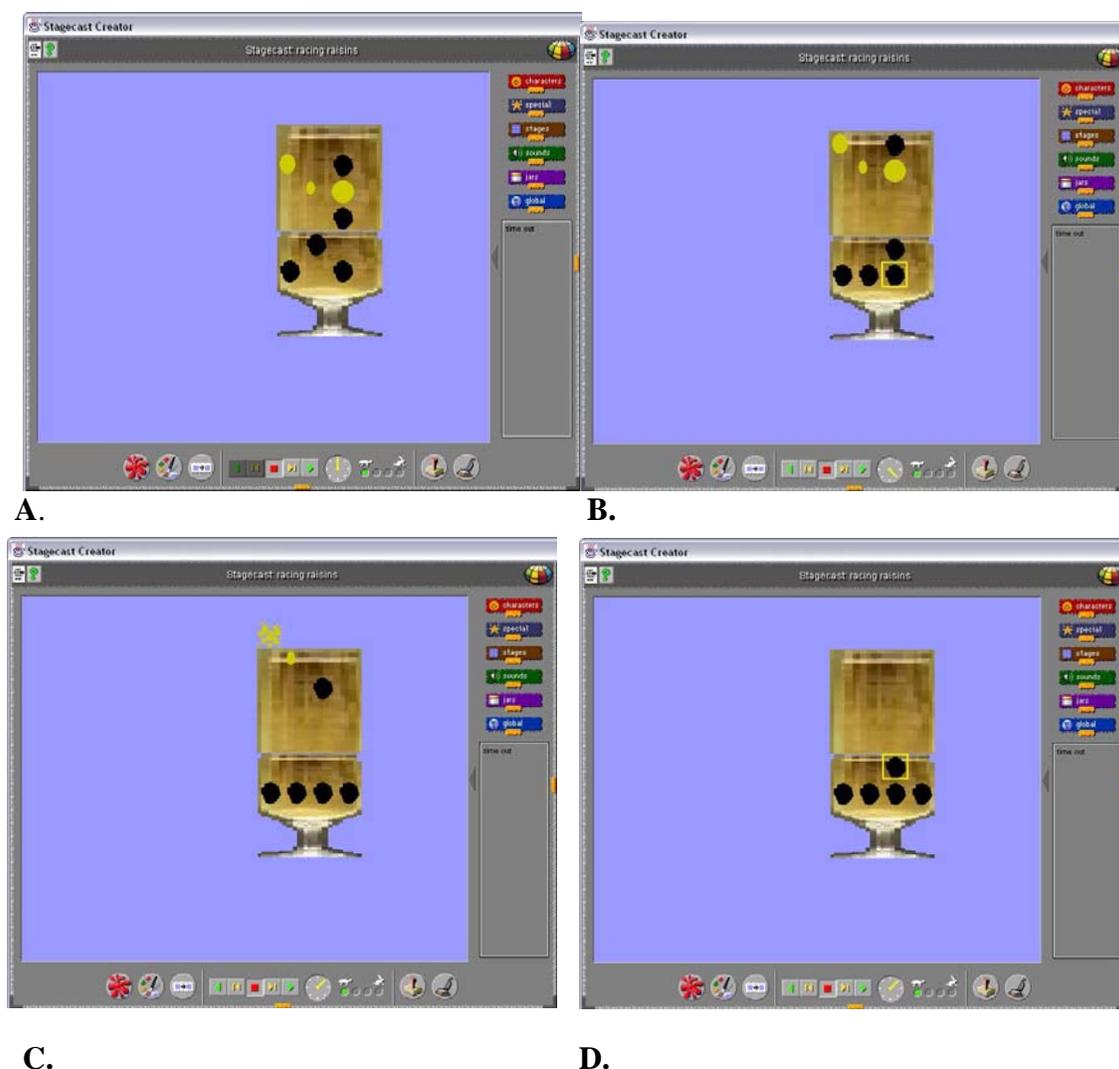


FIGURE 4.13. TEACHER CREATED CARBON DIOXIDE EFFECTS ON RAISINS IN GINGER ALE

Three more sessions were planned in order to create more simulations, but the teachers' schedules became too hectic and the sessions were cut short. However, even though there were only six sessions, a tremendous amount of information was gleaned from these sessions. Teachers were comfortable with the guided exploration card style of the minimalist tutorial and were able to learn to create simulations with self-study of the tutorial materials provided. These sessions also revealed that teachers could identify good candidates for simulation topics. This was also helpful in exploring motivational issues for our teachers. They were motivated enough to complete two simulations, but were frustrated with having to create their own objects and would have preferred access to a library of objects to reuse. The facilitators helped the users import some images from the

web into their simulation, but created all the rules for their working simulations. This indicates that teachers are motivated to perform activities that are useful for their classrooms. With minimal training, the teachers created working educational simulations that would be useful in supporting their science class topics.

This research reaffirms that teachers would find a software curricular aid very helpful in the classroom. There was also more evidence of problems encountered with typical direct manipulation visual programming environments which were referenced during the further research to complete the system. Technical support will be provided for the prototype system, and the aim of this experiment was to ascertain whether the system will be robust enough for teachers to incorporate simulations into their science teaching, what if any impact it has on their students and what impact they believe simulations and this environment will have on science education.

Working with the teacher team in the creation of educational simulations revealed a great deal regarding the attitudes of teachers toward programming, as well as their motivations, time constraints, and willingness to create simulation software if they feel that they and their students will benefit from the experience. Teachers are becoming much more sophisticated and are accustomed to having very good drawing facilities available to them as shown by their request for clip art. They felt that their efforts would be better used in helping to communicate the science and they were definitely trapped in the production paradox of knowing what they wanted to accomplish, but not wanting to spend extra cycles to learn how to create the desired affect. However, the teachers were conscientious enough that they continued working with the project until they had basic simulation projects working and had gained a good working knowledge of the environment. The aim of this study is to integrate this information into a packaged tutorial, and deliver it with the environment and pertinent examples to provide a wonderful educational simulation resource for teachers.

4.7 Implications of Requirements Analysis

The end-user population studied in this sample consisted of Montgomery County schoolteachers, in particular, teachers who were domain experts in the area of physical science, biological science, and physics. The age groups that teachers designed simulations for were middle and high school students. Each part of the requirements analyses added arguments to strengthen the design of the prototype system and supporting materials.

One of the main aims of this study is to increase the usability of simulation environments to increase levels of teachers' adoption of educational software environments. Creating this taxonomy of simulation environments enabled the researchers to become familiar with common simulation environments and identified environments that have a decent level of usability. This task also narrowed the field of candidate environments to four, based on the criterion of not being too cost prohibitive, supportive of a construction kit style of simulation creation and/or providing support for novice programmers with programming by demonstration. Based upon this preliminary study on learning to use the environments, the users' initial experience and reactions to an environment exert a large influence on their future use and adoption of the tool.

The initial informal usability inspection of the four environments led to a focus on three areas of interest: environment issues, drawing tools and rule creation support. Of the two environments with the most promise, AgentSheets offered a satisfactory environment, satisfactory rule creation facility, but needed work in the area of drawing tools, while Stagecast had a satisfactory environment and drawing tools, but needed to improve its rule creation facility. A more detailed inspection of the features and support provided by these environments was used to narrow the scope of the study. The preliminary study also revealed information that could be used to rank priorities for use during design and development in order to address high priority issues first, followed by the lower priority issues in turn.

Scenario & Claims analyses were used to directly guide design in an attempt to maintain or enhance upsides and mitigate or remove downsides. For instance, a positive claim like *support for renaming of objects* is very important for the support of the copy-paste reuse style often employed by novice programmers and this feature will be supported. Another claim, *robust drawing tools*, is also needed to support novice users and reduce their frustration with the environment and support for robust drawing tools will also be incorporated.

The Learning & Reuse study of the requirements analysis provided a benchmark for the summative Learning & Reuse study. The findings from this study fall into three broad categories: drawing tools and icons, rules and methods, and general environment. In the drawing tools and icons category, better drawing support was needed because most users became frustrated with the single pixel drawing tools. They required drawing tools that were more like paint, with a variety of pixel choices. Also users were very interested in the use of drawing and object templates such as clip art or objects that could be imported from the web. In the rules and methods category, the users found these facilities adequate for the creation of basic simulations, but encountered some logical interaction mappings that were not intuitive, causing them many errors, confusion and frustration. Many of the users simulations worked semantically, but the imposed syntactic structure of the environment caused many of these user errors. With the general environment, users were disoriented with too many windows. Six windows are generally needed to create even a basic simulation, and our novices were quite often confused and lost in the windowing structure. Some environment problems were caused by the use of GRRs, visual overlay problems, and the need for reuse support at various levels of abstraction. The introductory study revealed that user support within the environment with a pool of abstract objects would support template object creation and thus facilitate creation of functioning objects. This was a preliminary result and needs to be validated with further study.

The goals of the requirements analysis phase were to narrow the scope of work and to begin preparing a structure for the design and development of the prototype

system. Thus far, the preliminary study has revealed a number of inherent problems with the environments examined. The system proposed as a result will incorporate an environment for improved drawing facilities, a robust rule and behavior creation toolkit, a minimalist instructional tutorial to guide the learning experience, and support for reuse of simulation microworlds and their components. This completes our requirements analysis and these findings will be utilized for SimBuilder Development.

CHAPTER 5. SIMBUILDER DEVELOPMENT

This chapter contains details of the system design and refined requirements, which are both products of the initial requirements analysis. The details of the SimBuilder prototype system that was created are also presented. The system design was refined and captured with Unified Modeling Language (UML) as an object-oriented design standard for capturing requirements. This is followed by an overview of the language and programming environment that was utilized to complete this work, and the chapter concludes by presenting the high-level implementation details of SimBuilder.

5.1 System Requirements

The Statement of Purpose for this design was to create a system that will support teachers in their classroom activities as a curricular development aid. This system was dubbed Simulation Builder for Teachers or SimBuilder for short. As a general description, the system incorporates an environment for drawing, behavior creation, and reuse of simulation microworlds and their components.

Preliminary results and artifacts from Phase I of the methodology, the requirements analysis, were used to refine the initial requirements that were utilized both in the design of SimBuilder and in guiding the analytic and experimental evaluation of SimBuilder reported in Chapter 6 Phase II of the methodology (SimBuilder development) applied the refined requirements to create a high-level design using case activity diagrams. This design was implemented in the Smalltalk Squeak (Guzdial, 1999) environment and its major components and rationale are briefly described. During prototype development, informal usability evaluations (in this case, via demonstrations to colleagues and scenario walk-throughs) were used to fine tune the design of its functionality and user interface.

5.1.1 Phase II: SimBuilder Development

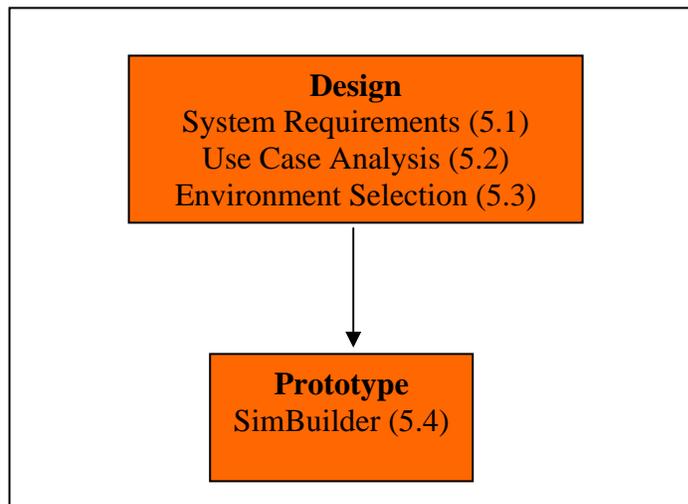


FIGURE 5.1. SIMBUILDER DEVELOPMENT

The results of the analytic and empirical studies provided the design requirements for a new visual programming environment that will support programming by the teacher community, and provide a framework for reuse. The claims from the analytic inspections and scenario-based studies, along with the information gained from the preliminary empirical studies, were analyzed to provide requirements for a new visual end-user programming environment aimed at science simulations and intended to mitigate the usability problems identified in the studied systems. These preliminary studies clearly showed that teachers need more control and input to the software that they will use and, at the same time, make the software must be as straightforward and engaging as possible. The first step was to compile the list of requirements gathered from the analytic and empirical evaluations. Table 5.1 offers a brief listing of the refined system requirements that were used in structuring the design and development of the new system. These requirements have been organized into categories related to the pervasive research questions concerning the support of learning and reuse in teacher simulation environments.

Table 5.1.

Basic System Requirements

Basic System Requirement
<p><i>I. Support for Learning</i></p> <ul style="list-style-type: none"> • Provide a rich set of drawing tools that allow the user to draw • Support simple undo and incremental testing • Incorporate clear, recognizable, common sense icons • Easy creation/execution of graphical simulations. • Support user creation of object behaviors/interactions • Support incremental testing of object behaviors/interactions • Satisfactory level of usability for novice programmers. <p><i>II. Robust support of reuse</i></p> <ul style="list-style-type: none"> • A base set of generic template objects that can be reused • Provide a rich set of drawing tools that allow the user to modify objects • Permit opening of multiple worlds to facilitate copy/paste and reuse learning. • Support creation of object behaviors/interactions with rule templates or toolkit • Include a library of reusable objects and simulation projects • Platform independent implementation (i.e. Java or Smalltalk) <p><i>III. Some secondary requirements</i></p> <ul style="list-style-type: none"> • Import graphics • Import background graphics • Support for multimedia and interactive simulations • Ability to use macro recorder to record actions and create rules.

The rationale for the support of learning requirements is as follows. A key task for novice simulation builders is the creation of simulation characters. This involves the visual design of a character's appearance along with the programmatic construction of a character's behaviors. The requirements thus are to provide a rich but easy-to-use set of drawing tools that will allow teachers to draw and modify objects, as well as to create simple object behaviors and interaction. The system must support simple undo, incremental testing, and a generally satisfactory level of usability, while simultaneously encouraging exploratory learning in the environment and continued use.

The rationale for the reuse-related requirements is as follows. In order to provide useful examples and convenient modification techniques, a base set of generic template objects will be offered, and the environment will allow the opening of multiple worlds to facilitate a copy-paste style of reuse. Teachers will also be able to create new rules from rule templates, including a library of reusable simulations and components. The target of our system is middle school teachers who have very little free time to create simulations from scratch. With the ability to reuse simulations, teachers can share work, plans and ideas, and thus be more productive.

Secondary requirements were also considered from the general perspective of making the environment richer and more convenient to use. For example, in the pilot tests, enabling users to import graphics, backgrounds, and to create multimedia and interactive simulations increased users' motivation and excitement in simulation development and use. All popular simulation builders and educational games have some facility to make simulations interact, and the more sophisticated examples also take advantage of the multimedia computers which are becoming the norm, particularly in educational settings.

The general rationale for building simulations as educational material was summarized in the earlier chapters and is very practical. Simulations are independent of time and place, which can provide a better conceptual model of a situation. Simulations can also be used to create virtual environments, taking learning into the realm of imagination (Kuyper, 1998). The emphasis on reuse is tied to the need for teachers to benefit from one another's work. Currently available systems have some low-level components that can be reused, but it would be preferable to be able to reuse higher-level components, up to the level of entire simulations. Finally, the system should encourage and support an on-line teacher community of simulation developers. The initial plan is to support reuse of simulations and their components at various levels.

The use of an object-oriented versus function-oriented approach will allow a more realistic simulation environment. Instead of having to use pseudo non-determinism hard

coded into the simulations, the objects will interact and respond to each other with message passing. This will allow a simulation to be a collection of objects that work collaboratively. Objects belong to classes and in an object-oriented paradigm classes can inherit features from other classes. This is an encouraged practice for expert reuse, to speed program development (Hume & Stephenson, 2000). While creating this environment for physical science teachers, a pattern or framework for educational simulations was identified that can be utilized by developers to populate this environment for other content areas or reuse the framework to create new educational simulations.

A complementary set of requirements is related to the learning materials that will be offered in support of teachers' learning and reuse of the environment. Tutorials will be created using the Minimalist model of instruction and these training materials will be included as a part of the system. Minimalism emphasizes the streamlining of instructional materials, and a task-oriented approach to determining and organizing the content of manuals, tutorials or other educational artifacts (Carroll & Rosson, 1990). The goal is to support users in accomplishing real and meaningful tasks quickly, and to allow them to take advantage of their existing task knowledge in learning about a new system.

5.2 Use Case Analysis

In creating SimBuilder, the research goal was to create a system that will meet a minimum set of usability requirements in a tool to support novice teachers in their creation of simulations. To achieve this goal, support for basic tasks such as the creation of a simulation, adaptation of an existing project and the reuse of a simulation must be provided. These three general tasks provide a context in which to explore the system design by building use cases that document the general and specific functions that will be incorporated into this environment.

Detailed analysis and design of the system was accomplished through an object-oriented decomposition of the system using Unified Modeling Language. The analysis begins with use cases, which describe the system in terms of functionality. A use case

analysis involves reading and analyzing the specifications, as well as discussing the system with potential users of the system (Eriksson & Penker, 1998). The actors in the SimBuilder system are identified as Teachers and Students. The Teachers are users of the system that interact directly with the system and use it to play, create and modify simulations. The Students are users of the system that, according to teachers' discretion, will indirectly and at times directly interact with the system, using it initially just to play simulations. However, with training from Teachers, Students may begin to experiment as they create and modify simulations. The Student actor will not be included in the use case diagrams, but at Teachers' discretion could perform all of the same activities as the Teacher actor.

Use Case 1: Creating a Simulation.

The use cases in SimBuilder for “creating a simulation” are defined as follows:

- Create New Simulation
- Delete Unneeded Rules
- Modify Existing Simulation
- Specify New Object Behavior/Interaction
- Test Object Behavior/Interaction
- Save Simulation
- Test Simulation

Note that an important component of the creation task is the option to modify existing materials, which is inherited from the general requirement to support rapid learning and construction through the reuse of suitable examples. The SimBuilder system analysis is provided as a use case analysis of Create A Simulation and is illustrated in Figure 5.2.

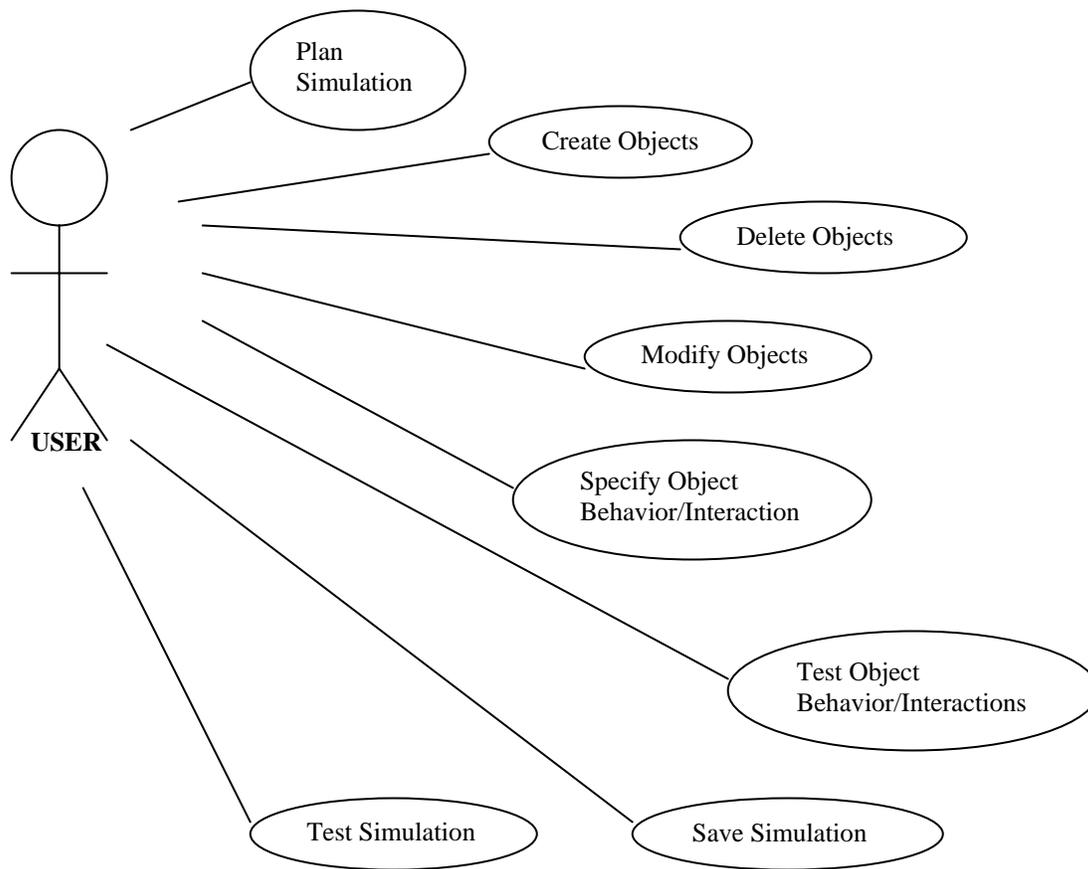


FIGURE 5.2. USE CASE DIAGRAM FOR CREATE A SIMULATION

In an effort to create an environment that would be helpful for teachers, a major consideration was to incorporate facilities that would encourage and support reuse activities. In addressing this concern, the researchers brainstormed activities and scenarios that teachers might engage in when they are initially introduced to the environment. One of these activities has been elaborated to further explore the option to reuse materials when creating a simulation.

A Teacher who is reusing material begins by opening the SimBuilder environment. The Teacher would look at examples that are available by browsing through the available Sims, then choose a near-neighbor candidate simulation that contained most of the content that they would like to use for a particular lesson. They would then study the simulation and find parts to reuse, after which there would be two scenarios of user action.

The following is a detailed example of a reuse scenario to illustrate the “adaptation of existing project” or “reuse” use case.

Jean wants to build a model of an ocean food cycle for her 5th grade students. The model should illustrate algae, plankton, and underwater plants growing as the result of sunlight on the water, small fish eating the algae and plants, whales eating the plankton, larger fish eating the smaller fish, and so on. Fish that don’t get anything to eat would die after a while. Alisa noted the similarity between the ocean food cycle and a model they previewed earlier. Jean feels that she can get a head start by reusing the example model, so she gives reuse a try.

This scenario can be described using a high-level use case activity diagram that introduces general teacher interaction with the system and system toolkits. The activities that are available for the user to perform are defined as follows:

- Utilize Drawing Toolkit
- Utilize Behavior Creation Toolkit
- Search for Existing Simulation

- Utilize Template Object
- Utilize Testing Facility
- Utilize Multimedia Toolkit
- Utilize Web Access
- Utilize tutorial

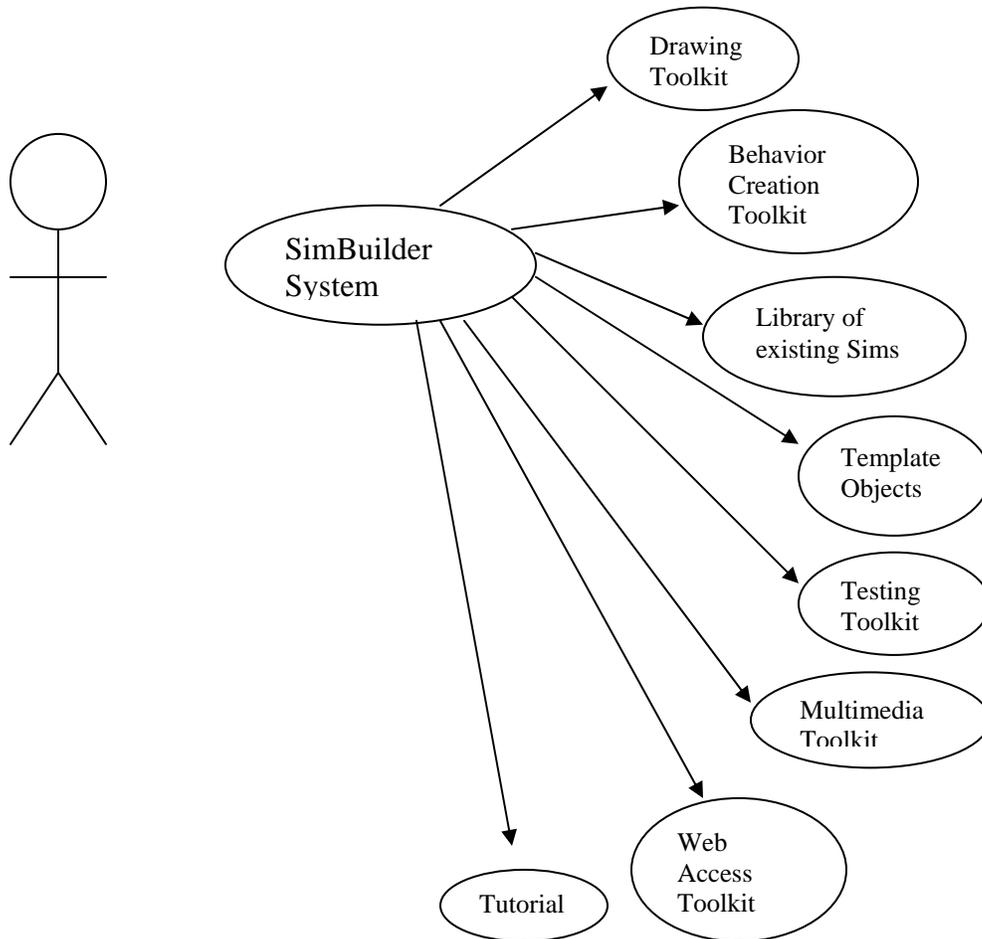


Figure 5.3. General Teacher Interaction With the System and Toolkits

Use Case 2: Adaptation of an Existing Simulation

If the Teacher chooses to modify a simulation, they must initially identify and study template objects and behaviors. Once the Teacher has studied examples, they will then begin to create new objects by using drawing tools to copy and modify template objects. After the Teacher has finished creating or modifying objects, they can make the simulation Web accessible. The Teacher may also utilize multimedia in the simulation. The final two steps in this scenario are that the user will test and save the simulation. The use cases in SimBuilder Use Case 3 for “Reuse of a Simulation” are defined as follows:

- Search for example, Simulation
- Modify or Reuse Simulation
- Delete Unneeded Objects
- Create New Objects or Modify Existing Objects
- Modify Existing Rules/ Interaction
- Test Object Behavior/Interaction
- Create New Simulation
- Test Simulation
- Save Simulation

The SimBuilder system analysis is provided as a use case diagram for reusing a simulation, as illustrated in Figure 5.4. The use case Modify or Reuse a Simulation can be described as follows:

1. If the Teacher has found a suitable simulation to reuse:
 - A. Extra objects will be deleted if unneeded
 - B. Teacher will create new objects if necessary
 - C. Teacher will create new rules or modify existing rules
 - D. Teacher requests that the system tests the objects behaviors/interactions

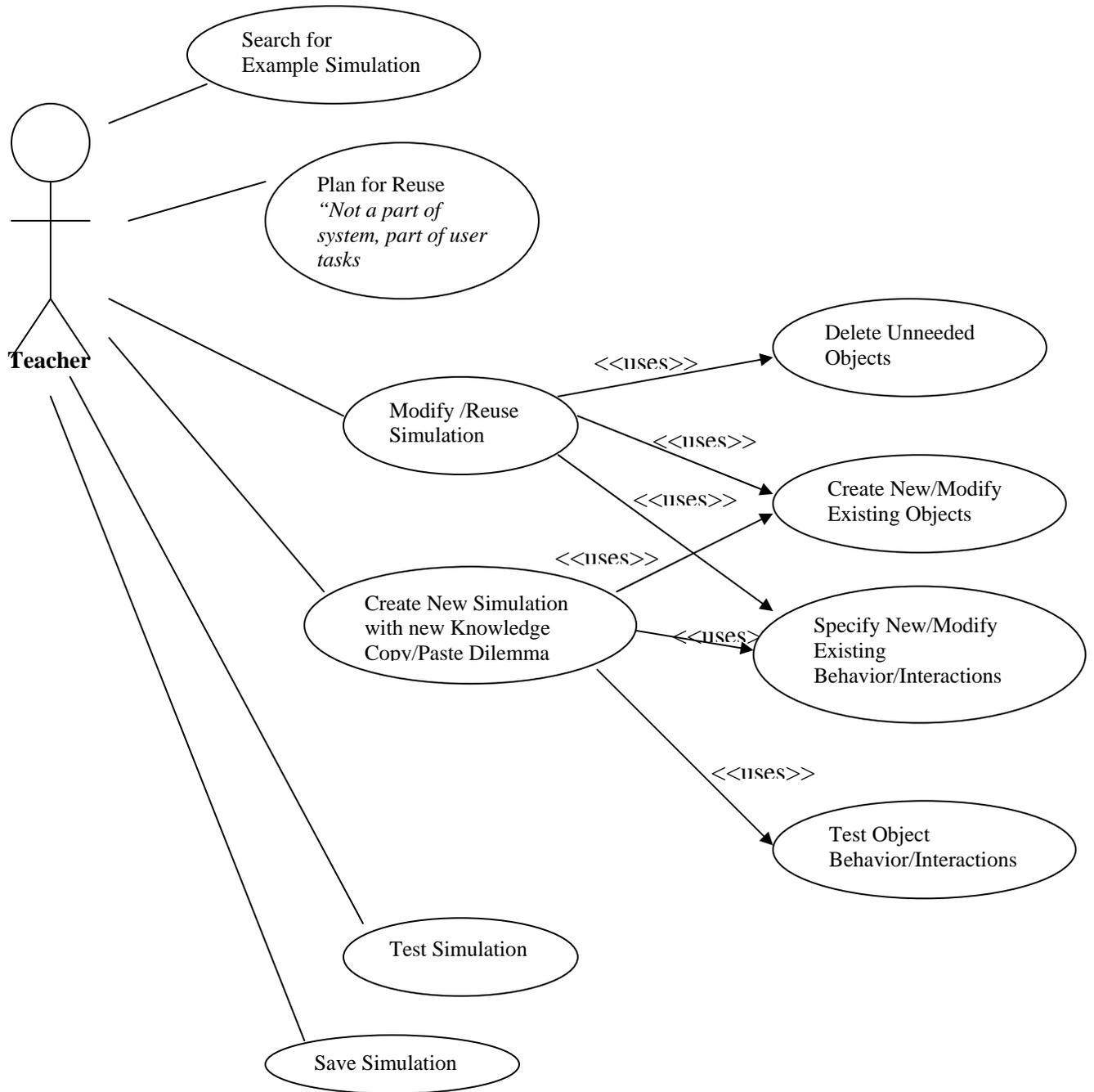


FIGURE 5.4. ADAPTATION OF EXISTING PROJECTS

5.3 Choosing an Object-oriented Language and Platform

In choosing a platform to use as a substrate for the simulation environment, many alternatives were investigated. With respect to programming language, the standardized procedural language C was used as the starting point as it combines elements of high-level languages with the functionalism of assembly language, so a programmer can manipulate bits and addresses. C is portable and supports the concept of data types, allowing almost all type conversions. The approach to writing most C programs is a functional top down approach, which works well for small systems with a small number of limitations, but it makes it difficult to reuse system components, for it is not easy to adapt elements to solve another problem. Because the study included a requirement for a sophisticated graphical user interface to support novice programming, an infrastructure that better supported visual programming was essential, which made the C language unsuitable for this project.

As an alternative approach to the classic structured-programming approach of the C language, the second system considered focused on Object-Oriented Programming (OOP). In modern software engineering, OOP is the dominant programming paradigm (Budd et al., 2002). It scales very well for large to small problems and most OOP languages include a large library of code to assist with development in various domains, which is very helpful in creating real world solutions to real world problems. In the real world, agents or objects interact to accomplish goals and this is consistent with the metaphor and computational structure assumed in OOP.

Objects interact by passing messages and functional languages communicate with procedure calls. In message passing, there is a designated receiver and the receiver determines the interpretation of the message. Usually the specific receiver will not be known until runtime, which is late binding between the message and the code or method, and gives programs more flexibility. This is in contrast to very early compile time binding in conventional procedure calls.

The basic characteristics of OOP languages are supportive of the types of functionality needed for this study. Kay (1993) summarizes these characteristics as follows:

- Everything is an object
- Computation is performed by objects communicating with each other to request that other objects perform actions. Objects communicate by sending and receiving messages. A message is a request action bundled with whatever arguments may be necessary to complete the task
- Each object has its own memory, which consists of other objects.
- Every object is an instance of a class. A class simply represents a grouping of similar objects, such as integers
- The class is the repository for behavior associated with an object. Classes are organized into a singly rooted tree structure, inheritance hierarchy. Memory and behavior associated with instance of a class are automatically available to any class associated with a descendant in this tree structure.

An object-oriented versus function-oriented approach should enable a more realistic simulation environment. Instead of relying on pseudo non-determinism hard coded into simulations, the objects in these simulations will be able to interact and respond to each other with message passing. This will allow a simulation to be a collection of objects that work in cooperation with each other. In most of the visual environments surveyed during requirements analysis (Phase I), realistic non-deterministic behavior (or randomness) was not directly supported. For example, to program an action so that it fires 66% of the time in the Stagecast environment, the user must program three separate rules. The rule that is to occur at 66% must be repeated twice because there is no option for an event to fire randomly; the third rule then fires 33% of the time. In other words, each character's rules simply fire exactly as specified. In contrast, by providing a random selection function, random behavior need not be approximated in this visual sampling paradigm. Instead, message passing allows a simulation to take place as a collection of objects that work collaboratively to produce the desired effect. This random

selection function mitigates the restrictions imposed by exact visual syntax in most visual programming languages.

Objects belong to classes and in an object-oriented paradigm a class can inherit features from its superclasses. Indeed this is an encouraged practice for expert reuse in OOP, to speed program development (Hume & Stephenson, 2000). The intent was to leverage this inheritance and reuse paradigm by identifying a framework for educational simulations that might be reused and elaborated by other technology developers, so as to expand the coverage provided by SimBuilder itself.

Based on the pilot studies, requirements, scenario and claims analysis, use case design, and investigations of several systems, the optimum programming environment for the creation of SimBuilder for Teachers was identified as Squeak, written in Smalltalk (Guzdial, 1999, www.squeak.org). Smalltalk was the first modern object-oriented programming language (Goldberg, 1984). It implements all of the functionality inherent to computers today, including bitmap displays, icons, windows, and mouse pointers. Windows, UNIX X-Win and Mac OS all have their roots in Smalltalk. It is a pure OOP language in that all computation in Smalltalk is modeled through objects and message passing, even primitive arithmetic and logic operations. As a full-featured OOP language, it also supports abstract data types, inheritance, and dynamic binding.

The Squeak user interface platform provides a multipurpose application programming interface (API), designed to encourage text input, graphical object creation and manipulation, code generation and testing, and information storage and retrieval. The version of Smalltalk used here includes typical system utilities and interactive tools (compiler, debugger, text editor). Smalltalk is an interpreted language; if syntax or other compilation errors are detected, they are reported via a pop-up text window with the option to use the debugging tools for immediate correction. Once an error has been corrected, the text is re-compiled into active object code and linked into the system, enabling the system to be continuously “running”. New code is tested by typing an

appropriate expression, selecting it, and issuing an evaluate command. This process is immediate, with no requirement for editing, compiling, filing, or executing “modes”.

5.3.1 The Environment

The original Smalltalk programming environment consisted of a system workspace window (with right-side scroll) where the user inputs code, a system transcript window (also with right-side scroll) where the results of the code are sent, and multiple menus that can be accessed with the pointing device, the three-buttoned mouse (i.e. red, yellow, blue). All of the commands of the system can be accessed through these mouse controls and their mappings are described in Table 5.2 (Goldberg & Robson, 1983). Many Smalltalk commands are close to language structure, therefore for a command to print the user selects “print it”, and for an expression to be evaluated, selects “do it” (Goldberg, 1984).

Table 5.2.

Smalltalk Mouse Buttons

Button	Left	Center	Right
Also Known As:	Red	Yellow	Blue
Windows 2 Button Equivalent	Left Click	Alt-Left Click	Right Click
Mac 1-Button Equivalent	Click	Opt-Click	Cmd-Click

Just as many features of the original Smalltalk environment were influential in modern user interfaces, the Squeak user interface has advanced the boundaries of today’s multimedia computers. Alan Kay (often referred to as the “father of OOP”) was one of the creators of Squeak. His team argued that some of the modern Smalltalks had strayed from the idea of a true Dynabook “a personal computer for everyone.” Thus, they created Squeak to re-emphasize Smalltalk’s original commitment to more personal dynamic media (Guzdial, 1999). Squeak is an open source project and freely available on the Internet, with active discussion lists about development and use in teaching to make it easy to share tips and work collaboratively on code (www.squeak.org). The Squeak license from Apple allows Squeak users to create anything they want as long as they

release it back to the network. Another important feature of Squeak is that it has been implemented with cross-platform goals and portability in mind, currently supporting all the following operating systems:

- Windows 98/95, NT, XP, and CE
- Macintosh
- Unix
- Be OS
- NeXT
- DOS
- Acorn RiscOS
- OS/2
- Squeak on many PDAs
- Zaurus PDA
- DEC Itsy PDA

As a modern descendent of Smalltalk, and an active project within the open source community, Squeak is both richer and more robust than Smalltalk and researchers, professors, or motivated students can examine, test, or change any part of the system, including the virtual machine itself, using only the Smalltalk language (Guzdial, 1999):

Squeak didn't happen by itself. It's the product of years of work by a core group of Squeakers namely Alan Kay, Dan Ingalls, and Ted Kaehler who met at Xerox PARC in the early 70s and worked together for many years. Scott Wallace worked at Apple on the LISA, and then worked with Dan on Fabrik (Dan's iconic programming system), then with Alan's 'Vivarium' school project. John Maloney was a 'graduate student of a graduate student' of Alan Kay's, was a principal member of the SELF group at Sun, and then joined Apple to work with Alan's Learning Concepts Group. Andreas Raab and Mike Rueger were graduate students at the University of Magdeburg in East Germany. They found Squeak on the Internet and Mike encouraged Andreas to do Windows and NT ports.

Andreas joined the group in '98 and Mike followed shortly after. Kim Rose joined Alan's Vivarium project in 1986 at Apple and has work on educational projects. Pat Brecker is an assistant to Alan and Kim and joined the Squeak Team in November 1996. (www.squeakland.org)

Many have sung the praises of Squeak as a GUI toolkit:

Don't let the bright, colorful GUI or experimental nature of Squeak's design fool you; it is not a toy. It is a complete, robust Smalltalk development environment. It comes with a powerful graphical framework called Morphic; a 3D world builder called Alice; Web and E-mail clients; real-time sound and speech synthesis; a full suite of development tools for the Smalltalk language and environment; and a lot more. Squeak is so powerful that it's used to write its own virtual machine. A translator converts the Smalltalk into C for optimum performance, but the next Squeak VM is written, tested, and debugged using the existing one. Having been co-created by Alan Kay, the man who developed the Smalltalk language and laid the foundation upon which modern graphical user interfaces (including the Macintosh and Microsoft Windows) would be built, Squeak also offers a glimpse into the future of human-computer interactions. Both Apple and Disney have used Squeak to prototype multimedia applications with innovative interfaces. (Read, 2002).

As an active open source project, it is not surprising that the Squeak project has undergone many revisions during the 2 years of SimBuilder development. The environment is distributed as a combination of "image", text source code, text change file, and an executable virtual machine (VM) file (see Figure 5.5). The environment can be updated either by filing in and compiling a set of changes (incremental or developer-shared modifications), or by distributing a new version of the image and/or source code (i.e., a new release).

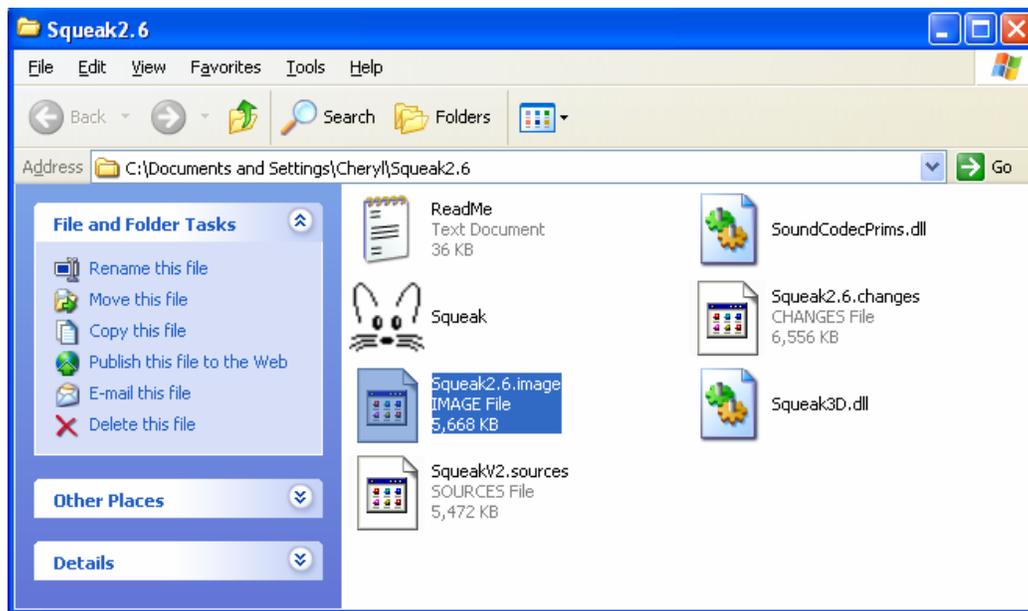


FIGURE 5.5. SQUEAK CONFIGURATION FILES

An early version of the Squeak user interface (Version 2.6) is illustrated in Figure 5.6. This interface is initialized with two windows that have the left scroll bar enabled (i.e. “Getting Started” and “Welcome To... Squeak”) and eight minimized “Play with me” windows. The Play With Me Windows contain the following projects: A collection of Sliders, ScrollBars and ListPanels; A group of forms (e.g. Polygons, Balloons Arrows, Curves); Morphic examples and examples of rotated text and writing on top of the another window (see Figure 5.7); an e-book containing the article Back to the Future: The Story of Squeak (i.e. a PDF reader with navigation controls); a project of multiple size text and as the form size changes the amount of words in the form change to fill its new size (See Figure 5.8); system organization chart; Squeak Alice (i.e. a 3-D creation environment; and a midi score player and piano roll.

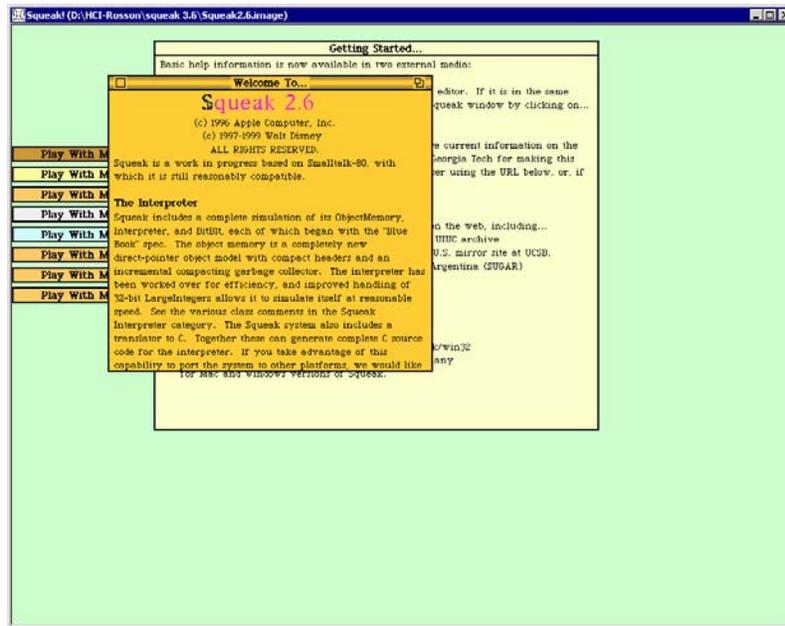


FIGURE 5.6. SQUEAK 2.6 USER INTERFACE



FIGURE 5.7. SQUEAK RULES EXAMPLE

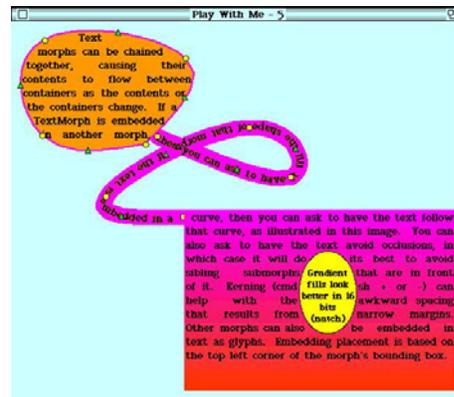


FIGURE 5.8. SQUEAK TEXT MORPHING

The Squeak 2.6 environment also includes support for creating interactive applications in two different modes—Morphic and Model View Controller (MVC). SimBuilder was constructed using the Morphic framework because most of the newest support for multimedia and novice creation is supported within this mode. For example, Morphic includes much more functionality directly available through context-specific

pop-up menus. If the mouse is placed over an object, a right click will bring up an object-specific menu; other menus are associated with the general environment. Figure 5.9 illustrates several useful menus: the World menu, the “Add a new morph”, “Authoring tools”, and “Playfield options”. For example, to add a new scripting area for morphs to interact, the option “authoring tools” can be used to access the menu choice of “new scripting area” (see Figure 5.10).

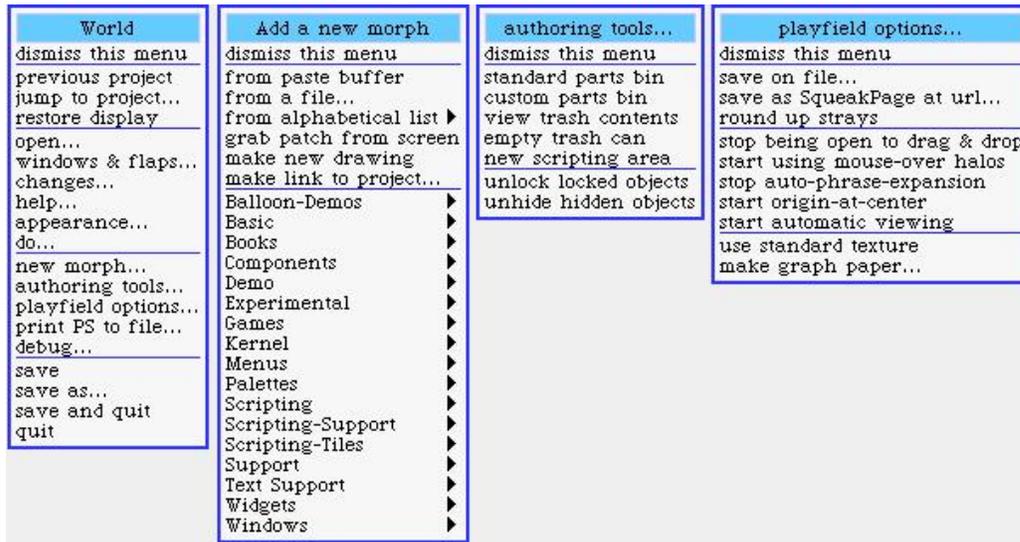


FIGURE 5.9. MORPHIC MENUS

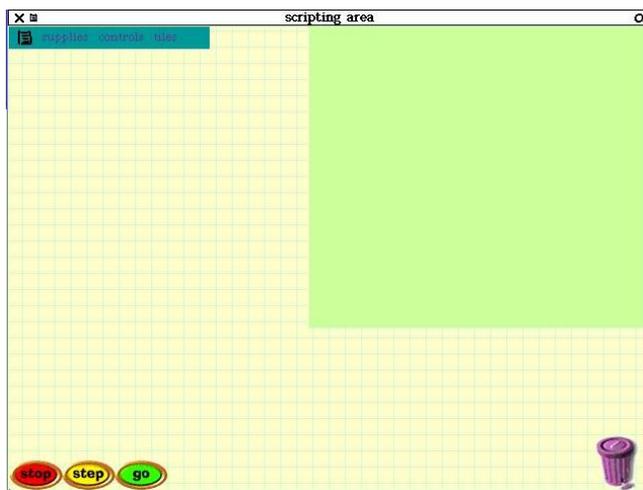


FIGURE 5.10. MORPHIC SCRIPTING AREA

The menus used to create morphs and give them attributes are all controlled by mouse options. The behaviors are organized in the following categories: basic, tests, color& border, geometry, motion, pen use and miscellaneous and each category pane has many rules where attributes are defined (see Figure 5.11).

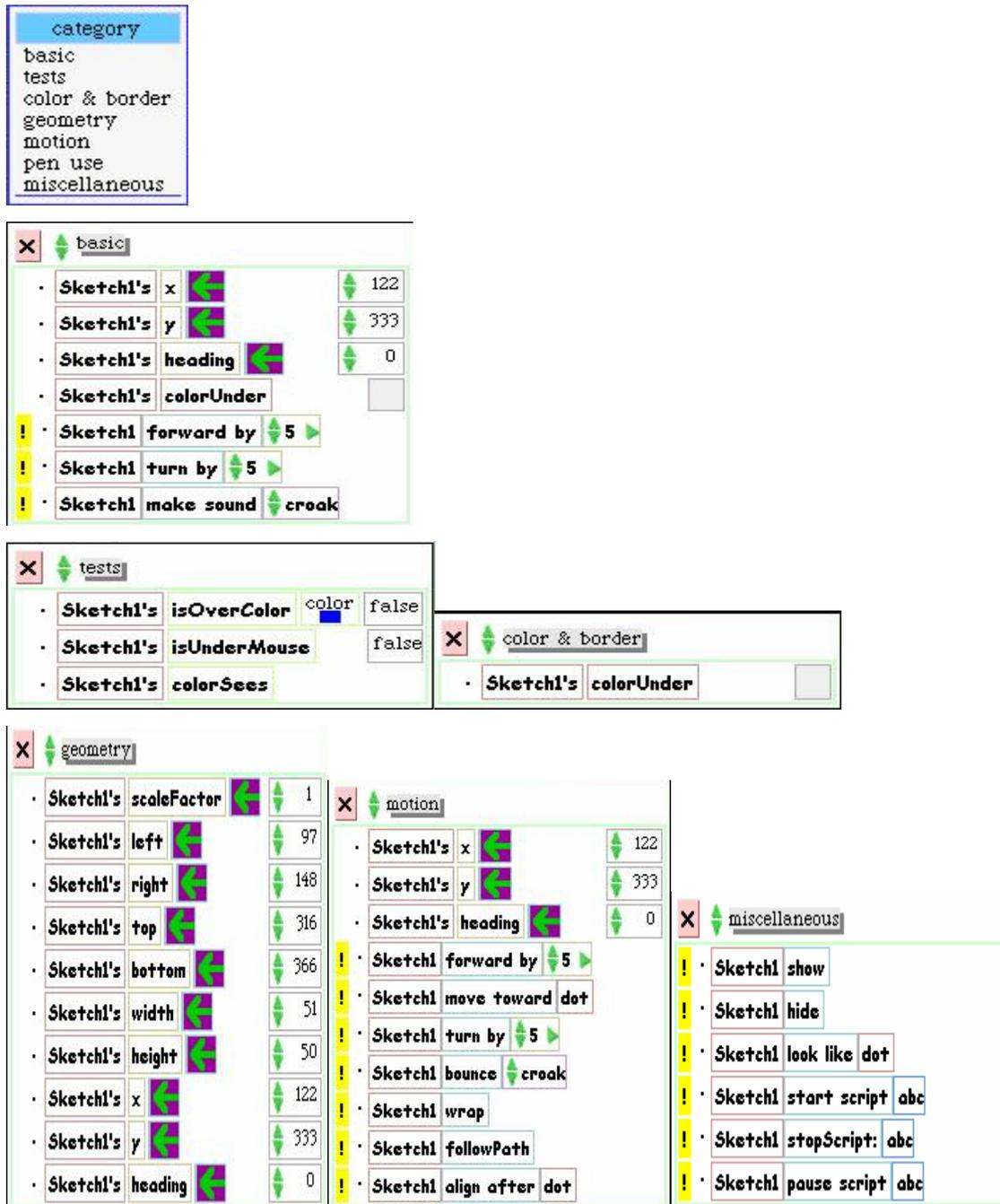


FIGURE 5.11. MORPHIC SCRIPTING CATEGORY PANES

A more recent version of Squeak (version 3.4) is initialized to open with two windows with left scroll enabled (i.e. “Welcome To... Squeak” and “The Worlds of Squeak” examples) and two minimized windows (i.e. “A Word of Caution” and “Readme.txt”) as illustrated in Figure 5.12. The Worlds of Squeak contains many of the project examples that were presented as “Play with me” windows in Squeak 2.6. This interface also has five helper “flaps” (or tabs) along the outside border of the system window: Squeak, Navigator, Widgets, Supplies, and Tools. The Squeak flap contains save, about, and trash can. The Tools Flap contains aids that a developer would use (e.g. method finder, system browser, transcript, workspace, pages, change sorter, file list, etc.). The Supplies flap contains multiple forms, buttons, sliders and an e-book. The Widgets flap contains objects that may be helpful in multimedia additions and simulation creation such as Scripting area, movie play, synthesizer, script player, and object catalog.



FIGURE 5.12. SQUEAK 3.4 SYSTEM INTERFACE

One new feature category that was particularly useful in SimBuilder development was the Navigator flap that contains button-aided navigation between projects, a “Publish It” button to save or share the project, and a paintbrush to draw new morphs (see Figure 5.13).



FIGURE 5.13. SQUEAK 3.4 NAVIGATOR FLAP

The differences in the Morhic Interface of Squeak3.4 that were not available or not as readily available Squeak2.5 improve the usability and support of the environment. The World menu has new options to load project from file, save project on file, undo, objects, projects, save as new version and two separate selections for windows and flaps. The Objects menu selection brings up the object catalogue that is stored in the widgets flap. The authoring tools category pane has been reorganized and the following options added: summary of scripts, remove all viewers, and objects catalog. Many changes were also made to the playfield options show and hide all players, fence enabled, auto-phrase expansion, automatic viewing, save to URL, etc. (see Figure 5.14 for authoring category and objects catalog).

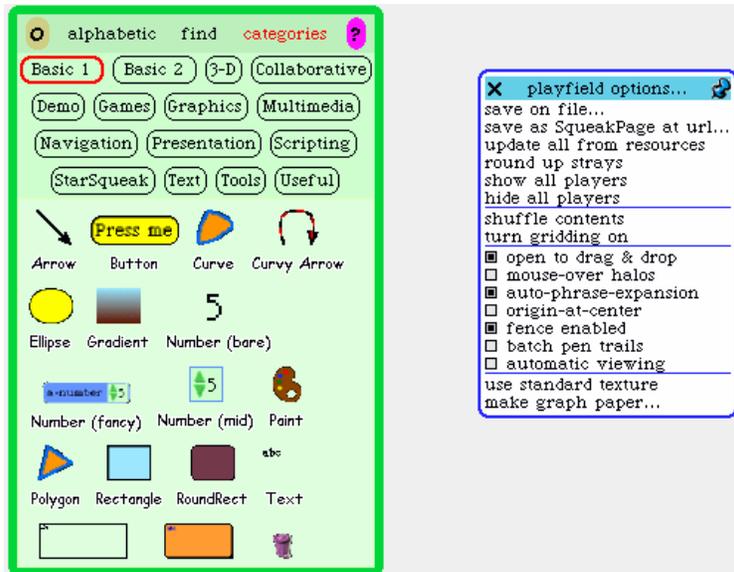


Figure 5.14. Object Catalog and Playfield options

5.3.2 A Scripting Example in Squeak

A simple example of building a user-controlled visual car illustrates the Squeak visual programming (Squeak developers call this “scripting”) facility, which is offered as an introductory tutorial available for novice users of Squeak. The starting Squeak environment is very basic, containing flaps that offer relevant supplies, navigation aids and resources to save projects. The project begins when the user constructs a visual representation of the car, using the provided paint kit (Figure 5.15).



Figure 5.15. Using Paint Kit to Create a Car

Once the user has created a visual representation of a car, they can press the “Keep” button to save the car. The next step is to give the object one or more rules, so that it can have visible behaviors. In order to operate on the car the user must invoke the car’s “halo” of handles (Figure 5.16); the blue cyan “eye” handle opens a view of the car’s current behavior, so that the user can see its properties and possible behaviors.

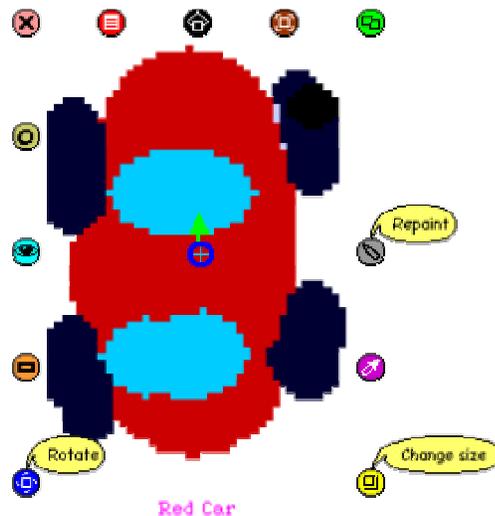


Figure 5.16. Car Halo of Operational Handles

The user can operate on each of the car’s rules individually, enabling them to see some of the functions of the car. For example, suppose the user wants to create a car that will move forward and turn by a number of degrees. The user must select the rule Red Car “forward by” and specify the number of forward spaces (in this case 5; Figure 5.17). To activate and test this rule, the user drags it from the behavior window and drops it on the background of the environment. The car will then move forward, but only in a straight line, and only for a distance of 5 spaces. If the user wants the car to do more than go in a straight line, they must add a second rule, Red Car “turn by” 5. With these two rules the car will appear to be moving in circles (i.e. moving for 5 spaces, turning 5 degrees, then moving, turning, etc).

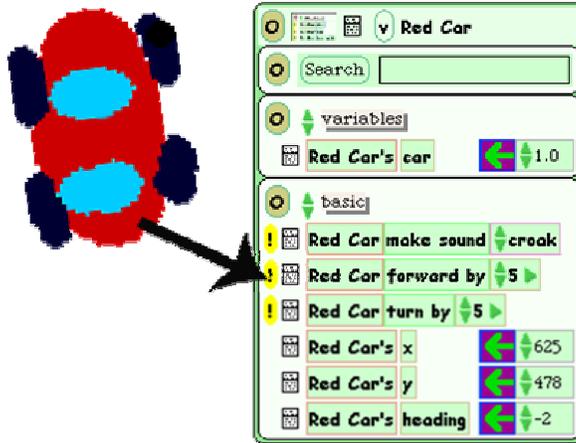


Figure 5.17. Behavior Script for Object to move forward

5.4 SimBuilder

Squeak was analyzed to determine how best to capitalize on the many features for designing user interfaces that are supported in the environment, while at the same time increasing the level of usability for a novice programmer audience. For instance, several features of the Morphic environment seemed to be problematic with respect to usability: The user must negotiate many modes for manipulating objects. The Left button and Right button clicks pick up an object, but if the user drags the object it can change the location of the dragged object from being on top of other objects to being underneath other objects. This can lead to problems with rules that seem logical to the user based on what they see, but because of the underlying spatial semantics can cause logical errors.

As another example, during execution the default for execution describes itself as “normal”, but in fact the user must change this mode for the object to function properly as a running simulation. There are a few extraneous objects in the environment that are used as work-arounds and/or place holders, but given how quickly this open source project is expanding (i.e. being constantly updated) this is not unexpected. Some object’s rules only fire as expected when the initial instantiation of an object is used and do not transfer to copied or cloned objects.

Initializing the movement of objects is somewhat complicated. The user must create a “moving” method invoke the object’s behavior viewer, select its rotate button, and manually change the forward direction of the morphic object. A few functions are missing in Squeak (e.g. delete) and some of the features did not work in an intuitive way (e.g. copy, set new object position at x or y position, duplicate). The environment also lacks a robust and convenient undo and erase. The object view flaps can be dragged off the viewable area and the interaction technique for repositioning the flap is poorly positioned. The system works very well for one object with a few methods, but once a more realistic simulation is created that includes several interacting objects, the system noticeably and proportionally slows down as the number of objects and processes increase. Even when an object is removed from the environment and placed in the trash the process is not actually removed; it still runs and will impact performance. The animation continuation of objects from one border to another in the playfield window (e.g. horizontal wrapping), should be handled by the window, but in early versions of Squeak was handled by the object. This requires the rule to be added individually to each object instead of being handled by the window.

The first SimBuilder prototype is shown in Figure 5.18, in the context of a simple Water Cycle model (based on the earlier studies with AgentSheets, Chapter 4). The SimBuilder simulation is comprised of fourteen objects or “morphs” that have behavior scripts; approximately half of these morphs have rules specifying their interaction. The Cloud morphs create rain, the Sun creates sunshine, and the Grass and Lakes produce Water Vapor.

In addition to the Water Cycle morphs, the SimBuilder environment offers morph template objects. These templates were added to the environment to aid the novice user, providing an easy mechanism to reuse generic objects rather than to start completely from scratch. For example, if a simulation user wants to modify the WaterCycle model, they can copy and modify one of these morph templates that already has encapsulated behavior or functionality analogous with its name.

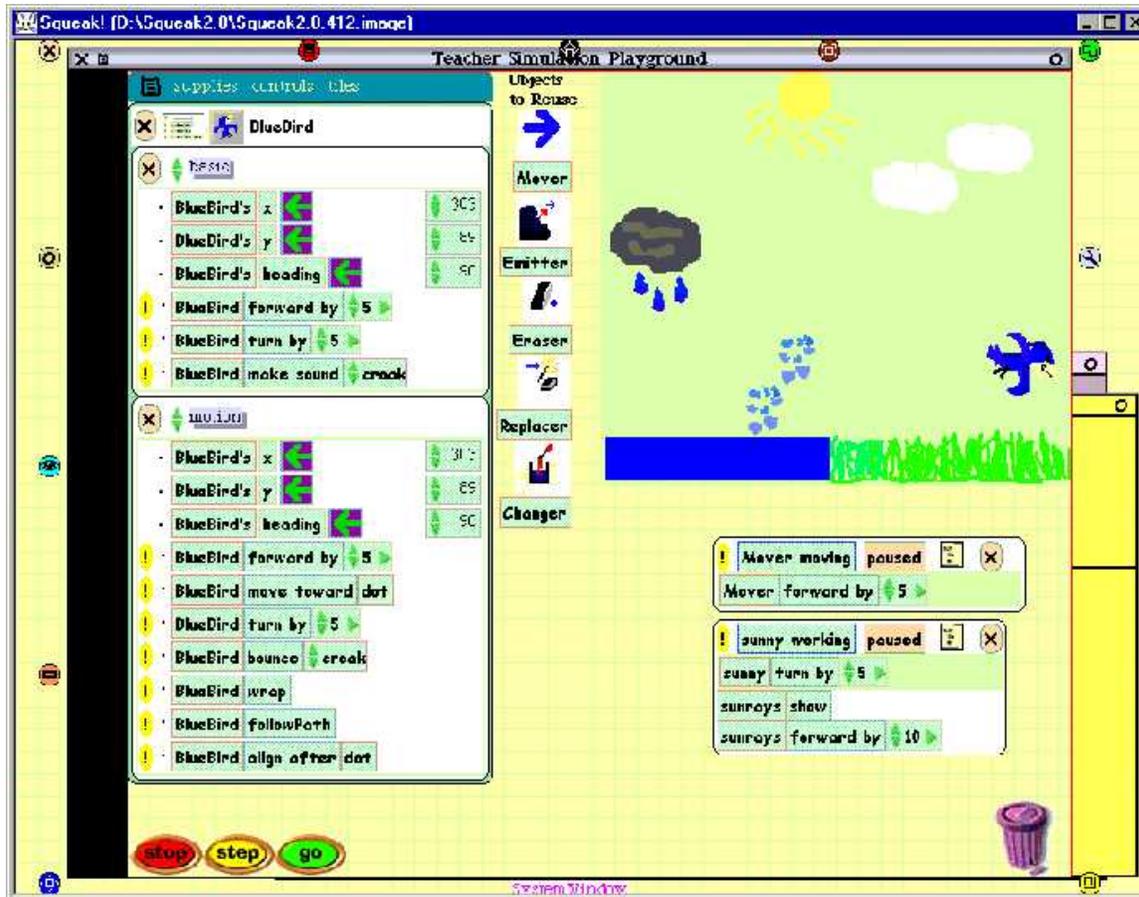


FIGURE 5.18. SQUEAK: WATER CYCLE SCRIPTING AREA

5.4.1 General Environment

A central problem identified for visual simulation environments during requirements analysis was that of too many windows being open at the same time, which caused users to become disorientated by the many interdependent windows. Complexity of this sort can add to a novice user's frustration and reduce their motivation to continue to use the software. To address this general issue, SimBuilder was designed to present a main interface organized as a simulation microworld where objects interact and there are at most one or two other windows that users manipulate to use the environment (Figure 5.19).



FIGURE 5.19. SIMBUILDER GRAPHICAL USER INTERFACE

Users start a simulation by using the scripting controls in the bottom left corner of the Playfield where the objects interact. If objects are not in the playfield they are not controlled by the scripting controls  , which allows objects not on the playfield to be in a sort of “timeout” mode where they can be used as templates or learning example objects. A paintbrush in the top right hand corner allows users to conveniently create and draw new objects for the microworld. The simple step of presenting these two functions in a readily available fashion for novices helps to scaffold the simulation creation interface for new users. They can carry out basic tasks of running simulations and adding objects to the simulation from the start. Providing this structure reduces the motivational and cognitive overhead of first learning how to interact with the creation mode of the environment, so that some activity is possible right away while novices become comfortable with the general concepts of visual simulation.

To undertake more complex operations, the user must assume the “developer” mode. As with Squeak, in SimBuilder all of an object’s behaviors are encapsulated as

rules within the object. Interaction with the object's behaviors is accomplished by selecting the object, invoking its halo, and operating on the object directly through the functions available in the pop-up menu (Figure 5.20). A few of the other operations that can be performed on the object include close, menu of more operations, pickup, move, duplicate, draw, recolor, rotate, resize, etc. (for details, see the last two pages of Appendix H, the SimBuilder Tutorial).



FIGURE 5.20. DARK CLOUD OBJECT AND HALO

Consider as an example the halo menu for the Dark Cloud object in Figure 5.20. In order to add active rules to this object, the user selects the cyan eye “Open a view of me” icon. This reveals all the scripting categories and rules that are available for this object. There are thirteen categories of scripts that are available for users, beginning with the basic category (Figure 5.21) that includes the object's ability to make a sound, object movement, and the object's position and heading. Users program with direct manipulation by dragging rules from condition and action templates to a rule window. To make a script active, the user selects a rule and drags it to the world, where it becomes an active script.

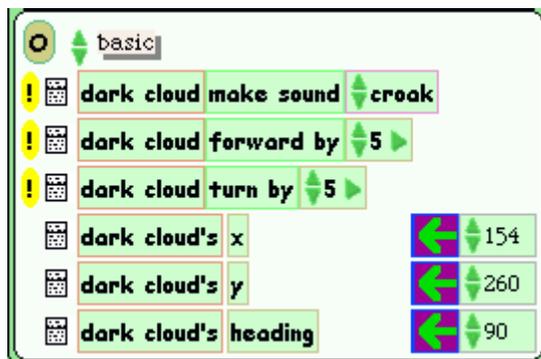


FIGURE 5.21. BASIC SCRIPTING CATEGORY

Rules can be individually tested with the “!”. This will test a rule and fire it once and the step control will step the entire simulation one cycle. When the simulation is running the entire playfield is highlighted in green; it is highlighted in red when the playfield is not in play. This coloring acts as a reminder for the user. In the first version of SimBuilder, a “wrapping” script was provided for morphs moving through the playfield, so that when a morph hit a left border it would automatically appear at the right border (this has been provided as built-in functionality in more recent versions of Squeak). Template Morphs were added to the environment for users to copy and paste functionality. The Template Morphs “Objects to Reuse” were as follows: Mover, Emitter, Eraser, Replacer and Changer. Each one of the templates contains one category of use studied during requirements analysis and selected as promising for use as primitive components in simulation creation (Chapter 4). For example, a Template Morph that has been particularly useful for novices creating and programming simulations is the Emitter. This example can be reused to create any object that itself creates another object (e.g. a cloud creating rain).

The current version of SimBuilder, which was used in the comparative analysis with AgentSheets as reported in the next chapter, opens with two windows with left scroll enabled (i.e. “Welcome To... Squeak” and “The Worlds of Squeak” examples) and three project models (i.e. WaterCycle World, Starter World and Ozone World), as illustrated in Figure 5.22. It also presents an initial Squeak Morphic project and scripting area. The scripting area contains many objects, all of which can be manipulated via Morphic halos, scripts or instance variables. In this prototype of SimBuilder, the system flaps have been removed to reduce the complexity of the graphical user interface.

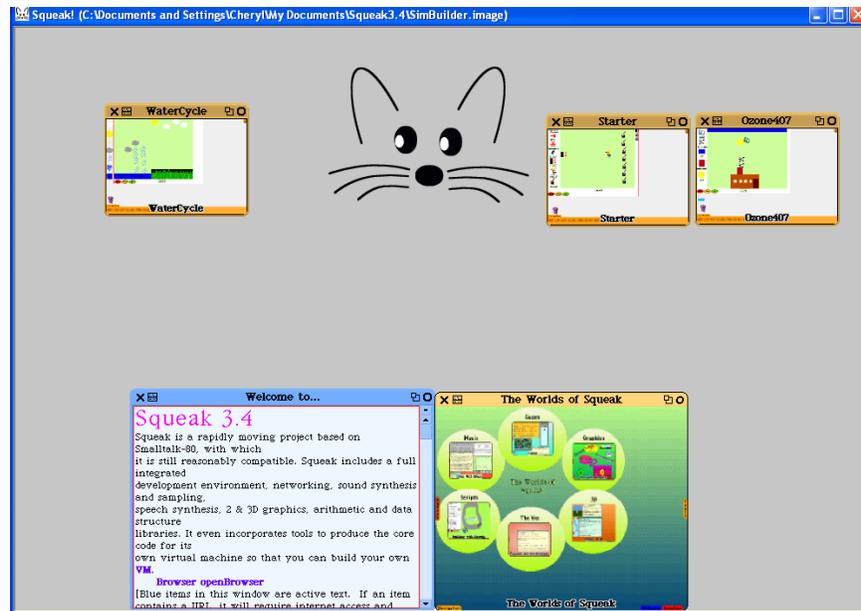


Figure 5.22. SimBuilder Prototype Interface

Other differences between the SimBuilder Interface and Squeak focused on usability and support of the environment (as for the example of wrapping, some of which were implemented initially, but no longer required as enhanced versions of Squeak became available). Most of the system changes were organizational issues, changing the focus and accessibility of some features, and adding features to support creating more robust simulations than the current one or two objects working philosophy used by most of the example simulations created in Squeak. A few scripts were added to increase the functionality of the Align parameter (used during movement), for example,:

- Morph align before dot “Self goToLeftOf: Morph.”
- Morph align above dot “Self goToTopOf: Morph.”
- Morph align below dot “Self goToBottomOf: Morph.”
- Morph delete “Self delete: Morph.”
- Undo by repositioning and emphasizing the button command

Finally, SimBuilder provides one very important function—Reset simulation to starting state—in a very convenient and simple form as a button command.

We have introduced the SimBuilder environment. In Chapter 6 we will give details of the evaluation of SimBuilder vs. AgentSheets, and present the formal discussion and conclusions of this research in Chapter 7.

CHAPTER 6. PHASE III: COMPARATIVE EVALUATION

Chapter 6 presents Phase III of this work as the summative comparative evaluation of this research, created environment and supporting principles. The initial method of evaluation included analytic evaluations of AgentSheets and SimBuilder in the form of an informal usability inspection (Section 6.1). The second method of analytic evaluation was a qualitative evaluation and performed as scenario and claims analysis (Section 6.2). The next step in the evaluation was an empirical investigation, a comparative study of learning and reuse in AgentSheets and SimBuilder. In the empirical evaluation section (Section 6.3), we present methods for the work, materials used, experimental data (i.e. demographics, user satisfaction questionnaires), procedures, experimental metrics (i.e. artifacts, timings, etc.) and experimental observations. The empirical study concluded with a final analytic evaluation in the form of an expert user interface evaluation of artifacts created during the study (see section 6.3.5). The chapter concludes with a discussion of the experimental hypothesis and implications of the study. These results and implications will be used to support our framework for learning and reuse programming for novices.

Phase III

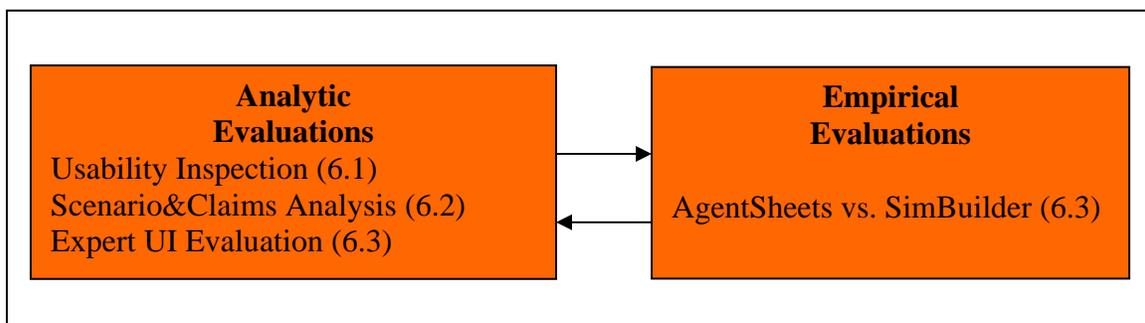


FIGURE 6.1 COMPARATIVE EVALUATIONS OF SIMBUILDER AND AGENTSHEETS

6.1 Informal Usability Inspection

As a means of documenting critical differences between key functionality and also differences between AgentSheets and SimBuilder, we completed a feature comparison as summarized in Tables 6.1 – 6.3. This work extends the informal usability inspection from Chapter 4, Tables 4.2 – 4.4 to include a comparison of AgentSheets and SimBuilder. AgentSheets utilizes graphical rewrite rules to handle complex situations and equations. SimBuilder utilizes the Smalltalk computational paradigm where each object knows the messages that it can understand, and associated with each message is a method. There is a method creation kit or rule window where users can choose predefined methods implemented as a technique to increased ease of use for novices. With variable naming, SimBuilder is much more malleable than AgentSheets. If the user wants to reuse a character, s/he just has to copy it or literally change its name by selecting it. In AgentSheets the user is allowed to copy and clone agents, but the new agent's functionality must remain the same as the base class.

In AgentSheets and SimBuilder basic user functions are predefined, with a larger function set available in SimBuilder. Additionally in both AgentSheets and SimBuilder the sophisticated user can extend the construction kit with user-defined functions. Our analysis suggested that the SimBuilder construction kit allowed the user more flexibility and a larger variety of rule behaviors than can be created with AgentSheets. This aids the user by giving them added power to create more realistic simulations. There are thirteen rule creation palettes in SimBuilder as compared to three palettes in AgentSheets. The SimBuilder palettes or categories are the basic, scripts, color & border, geometry, motion, pen use, tests layout, drag & drop, scripting, observation, graphics, and miscellaneous, which allow the novice to create simulations by using direct manipulation techniques. The AgentSheets palettes are the condition, action, and triggers palettes, which also allow the novice to create simulations by using direct manipulation techniques. SimBuilder and Squeak also has a facility to support multimedia, animation, and computer supported collaborative work. SimBuilder has a more game-like character; it can accommodate multiple stages (e.g. a hard stage and an easy stage) of a game or simulation easily with a

links between worlds. Worlds can be traversed with a link and the navigation bar. AgentSheets connections are inherently tied to their visual representations, and spatial location.

In our prior informal usability inspection, during phase II of our research the requirements analysis, we inspected four environments and AgentSheets was the best of the studied systems. We rated it the best in areas of standard components, rule-based object behaviors, incremental testing, object copying, library of simulations, and web accessibility. In our comparative study of AgentSheets and SimBuilder, we looked at the state-of-the-art system AgentSheets and compared features relating to environment issues with the aim of determining which system based on our experience had the most promise for supporting novice programmers in their building of educational simulations. In the area of environment issues 11 of 21 cases SimBuilder showed some improvement over the AgentSheets system and in 9 cases they rated at an equal level of satisfaction. Our determination in this informal usability inspection is that the SimBuilder system has an advantage in features that deal with environment issues including visual editing of rules, rule-based object behaviors, realistic animation of objects, window management, standard components, cross-platform use, web accessibility, reuse-adaptation of objects, object destruction, object copying, saving projects or worlds, incremental testing, environment startup, support for classes of similar objects, and for these features SimBuilder ranked “Very Good” on the usability inspection scale (environment issues illustrated in table 6.1).

Table 6.1

Usability Inspection: Environment Issues

Environment Issues	AgentSheets	SimBuilder
Visual editing of rules	2	1
Rule-based object behaviors	1	1
Realistic animation of objects	2	1
Window management	3	1
Standard components (e.g. condition/actions)	1	1
Library of simulations	1	2
Default new object	3	2
Cross-platform use	2	1
Web accessibility and use	1	1
Simulation speed controls	3	2
Reuse-adaptation of objects	2	1
Object destruction	3	1
Object copying	1	1
Saving projects or worlds	1	1
Icon interpretability	2	2
Auditory feedback for actions	2	2
Incremental testing	1	1
Global variables for state change	2	2
Environment start-up	2	1
Multiple active projects	3	2
Support classes of similar objects	3	1

Usability Inspection Levels: 1. Very good 2. Satisfactory 3. Unsatisfactory

In the earlier requirements analysis AgentSheets ranked second best of the studied systems for drawing features; the higher ranked system for drawing was Stagecast Creator. If the comparison in this category were made to Stagecast, we predict that there would have been no perceived difference in satisfaction level for drawing tools, but since we were testing against not the best drawing tool but the best overall simulation development tool our comparison was made with AgentSheets. In 5 of 9 cases SimBuilder showed some improvement over the AgentSheets system and in 3 cases we rated them at an equal level of satisfaction. Our determination in this informal usability

inspection is that the SimBuilder system has an advantage in features that deal with drawing tools in the areas of flexible point size, undo, and flexible object size, and paint fill, such that SimBuilder ranked “Very Good” on the usability inspection scale. Interestingly, although one might expect that paint fill would rank the same on all environments, we observed that when utilizing the paint fill on SimBuilder the point where the fill is indicated is with a cross hair, and with AgentSheets the analogous point where fill is indicated is the end of the drop of paint. Our earlier work with AgentSheets had indicated that many users had problems related to filling the wrong area when using this bucket fill. Users seemed to assume that the bucket should be centered over the area where the fill was to occur (environment issues inspection illustrated in table 6.2).

Table 6.2

Informal Usability Inspection: Drawing Tools

Drawing Tools	AgentSheets	SimBuilder
Predefined drawing objects	3	2
Drawing canvas reset	2	2
Undo	3	1
Paint fill (e.g. paint bucket)	2	1
Block object fill	1	1
Flexible point size	3	1
Imported images	2	2
Desktop color sampler	2	2
Flexible object size	3	1

Usability Inspection Levels: 1. Very good 2. Satisfactory 3. Unsatisfactory

In the prior informal usability inspection AgentSheets was rated the best in areas of rule creation and rule priorities. When we compared AgentSheets with SimBuilder on features relating to rule creation, in 2 of 3 comparable issues SimBuilder seemed to have advantages over the AgentSheets system. We would need a more comprehensive study of rule creation to make a more meaningful comparison, but our initial determination in this informal usability inspection is that the SimBuilder system has a slight advantage in features that deal with rule creation in the areas of system extensibility, and rule priorities

where SimBuilder ranked “Very Good” on the usability inspection scale (drawing tools inspection illustrated in table 6.3).

Table 6.3

Informal Usability Inspection: Rule Creation

Rules	AgentSheets	SimBuilder
Graphical rewrite rules	1	NA
Programming by demonstration	NA	1
System extensibility	2	1
Rule priorities	2	1
Rule activation tracing	2	2

Usability Inspection Levels: 1. Very good 2. Satisfactory 3. Unsatisfactory

In drawing conclusions from these inspection results, it is important to note that the ratings were generated informally by the researcher in discussion with colleagues. No independent judges were trained or exposed to the systems. Also, the inspection results discussed in Chapter 4 contributed directly to the requirements analysis and design of the SimBuilder simulation environment, so it should come as no surprise that the new system is “better” than AgentSheets on many of the features summarized in Tables 6.1 – 6.3. Nonetheless, it is useful to re-visit system features at this level so as to document the ways in which SimBuilder appears to have improved on AgentSheets, at least an initial informal inspection.

6.2 Scenario and Claims Analysis

Another view of the design changes embodied in SimBuilder can be seen by analyzing scenarios of use and associated claims, contrasting these to the earlier analysis reported in Chapter 4 (AgentSheets versus Stagecast Creator). Table 4.5 and Table 4.6 document the continuation of the earlier scenario-based analysis, but now using the scenarios and claims to compare AgentSheets and SimBuilder. In this case our goal was to achieve summative evaluation goals, namely we wanted to document the hypothesized improvements in SimBuilder over AgentSheets. By recording claims, we also are able to

anticipate good outcomes of our interaction design and identify remaining interaction problems.

In this analysis we use the same two scenarios described in Chapter 4, taking into consideration the usability of the environment with respect to both immediate and extended use. The first task goal was to explore, learn to use the environment, and create a simulation, which is conveyed in scenario 1. The vision of reuse has important implications for reducing the effort of simulation construction, as well as for sustainability of a learning community of teachers. A reuse goal is reflected in the second task goal, which is to create a simulation utilizing reuse techniques that may be employed in the environments. The scenarios were as follows:

Scenario 1: Exploration and Learning (Environment Focus)

Scenario 2: Adaptation of Existing Projects (Reuse Focus).

6.2.1 Learning Scenario and Claim

In our efforts to study learning in these visual programming environments, we performed a scenario and claims analysis to identify issues that support their current design and implementation as positive claims, and to also identify negative claims or issues that may indicate areas that need to be redesigned or improved in either environment. In each instance we will give the general task context for scenario, detailed scenario with specifications pertinent to the described environment, and conclude each section with the claims analysis and their implications. The task context for scenario 1 (a teacher learning to build a simulation) is illustrated in Table 6.4.

The fully detailed scenario instantiating this general scenario for the case of AgentSheets learning and creation is illustrated in Table 6.5:

Table 6.4

*Task Context for Learning Scenario*1) *Teacher creates new active objects for water cycle simulation.*

A teacher wants to learn how the environment works and how to do basic creation tasks. The teacher will investigate a water cycle model and once she is comfortable with how the simulation operates she will add her own new active object into the water cycle model.

To begin the learning session the teacher opens the environment and runs the simulation. She will investigate the behaviors of objects within the simulation to ascertain what behaviors cause their actions. Once she has a basic understanding of the actions taking place and how to modify the look of an object and the behaviors of an object, she will venture to add her own new object into the simulation. Her addition is a bird that will fly through the simulation. She will run her modified simulation and debug it until her bird has the desired behavior.

Table 6.5

*User Interaction Scenario Describing AgentSheets Learning Activity*1a) *Teacher creates new active agents for water cycle simulation using AgentSheets.*

To begin the learning session the teacher opens the environment by selecting the AgentSheets icon, the environment and splash screen open. Next the user has to open a project to get started and has to open a worksheet that is associated with the project in order to see the simulation. Now she runs the simulation. She will investigate the behaviors of objects within the simulation; the user will select an agent, press the button edit agent to see the rules that are currently available for the agent.

Once she has a basic understanding of the actions taking place and how to modify the look of an object and the behaviors of an object, she will venture to add her own new agents into the simulation. She will create by adding a new agent, the first step is to press "new agent" button, then input the predetermined size for the agent (the users has two choices of 16x16 or 32x32 pixels), next step is to enter the name of the agent, then the systems creates a template agent, the user must clear this AgentSheets logo from the template. She will then use the single pixel based drawing tool to create her bird.

To have a creation that interacts with the environment the user will now add rules to her creation so that her new addition is a bird that will fly through the simulation. The user selects new rule and receives a rule template, the user then opens the condition palette and action palette to create a new rule. She creates her rule through direct manipulation techniques and drags one condition to the precondition holder and drags one action to the post-condition holder. The user also has to the select the direction of the movement rule by pressing on the direction selection button and using direct manipulation to select the direction of flight. Then she applies the rule for it to take affect. Then the user tests her new creation and it should fly successfully.

Details of the associated claims for this AgentSheets learning scenario appear in Table 6.6. The user is not satisfied with having to create the bird one pixel at a time and wants to use clip art, or use an object from the web instead. In one case the user just scribbled the name or initials of the object instead of having to create it. Next the user had a hard time adding a bird to the worksheet because the brush that is used to draw the bird is the same icon used to add the bird to the worksheet. This may have been intentional in the design to reduce amount of icons, but was confusing to the user. The user then added rules to her bird to move east, but this was also a little confusing to the user. The semantics of the language ask if the bird sees itself in the here position then the bird should move forward. Adding this extra stipulation was confusing to users, and also finding the direction selector was not intuitive. If they thought their rule worked logically, it usually did if this precondition had been omitted.

Continuing the analysis, we shift attention to the SimBuilder environment, working from the SimBuilder learning scenario presented in Table 6.7:

Table 6.6

Claims Analysis of an Exploration and Learning Scenario: AgentSheets

Usage Feature	Possible Upsides (+) or Downsides (-) of Feature
Usability of drawing tool	+ single pixel based drawing pen for detailed object
	+ single pixel based eraser
	+ undo of last action
	+ easy clear
	+ block fill
	+ paint bucket fill
	+ eyedropper to pickup a color
	+ easy to add many objects to workspace as user desires
	+ pseudo random rule firing with percent chance
	- BUT pixel size drawing tool was frustration for users and doesn't give them the variety of tools they are accustomed to with most paint applications
	- BUT size restriction only allows two size objects
	- BUT eraser is only single pixel size
	- BUT undo history only stores one past action
	- BUT bucket fill causes user error because the point of fill is the tiny tip of paint drop
- BUT user frustrated in figuring out how to add object to workspace, same icon for drawing and placing object in workspace was confusing	
- BUT does not support random behavior	
Usability of Rule Toolkit	+ easy rule creation from construction kit with direct manipulation
	+ large selection of rules
	+ easy to add behaviors as rules
	+ easy rule testing
	+ rule test has color highlight to indicate which rule fired
	- BUT user gets confused with visual syntax of rules in particular the direction parameter
	- BUT semantics of rules may be confusing (e.g. extra precondition of direction parameter that the object sees itself in the here position)
- BUT does not address random behavior (e.g. the user would indicate what percentage of the time an event will happen)	

Table 6.7

User Interaction Scenario Describing SimBuilder Learning Activity

1b) Teacher creates new active agents for water cycle simulation using SimBuilder.

To begin the learning session the teacher opens the environment by selecting the SimBuilder icon, the environment, welcome window and simulation windows are in view. Next the user clicks a project to open it and get started. Now she runs the simulation. She will investigate the behaviors of objects within the simulation; the user will select an object by double clicking to see the rules that are currently available for the object.

Once she has a basic understanding of the actions taking place and how to modify the look of an object and the behaviors of an object, she will venture to add her own new objects into the simulation. She will create by adding a new object, to do the user selects the paintbrush and begins to draw her new object. She can then create any type of size bird she wants with nine choices of point size. When the user has finished her creation she presses "keep" when she is satisfied with the creation.

To have a creation that interacts with the environment the user will now add rules to her creation so that her new addition is a bird that will fly through the simulation. The user selects the object with an alt click to activate the objects halo and opens a rule viewer. This is a view to the rule toolkit. She creates her rule through direct manipulation techniques and drags one rule from the toolkit for it to take affect. Then the user tests her new creation and it should fly successfully.

Details of the associated claims for the SimBuilder learning scenario appear in Table 6.8. The user was satisfied with having a variety of drawing tools available to create her bird and all users create their renditions without complaint. The user did not have to worry about adding her creation to the workspace or to add rules for her creation to move. All the user has to do is to indicate which direction they want as the bird's forward direction by rotating its orientation arrow; once this is done the bird will move in the desired direction.

Table 6.8

Claims Analysis of an Exploration and Learning Scenario: SimBuilder

Usage Feature	Possible Upsides (+) or Downsides (-) of Feature
Usability of drawing tool	+ multiple pixel size drawing pen for detailed object
	+ any size object
	+ multiple size eraser
	+ undo of last action
	+ easy clear
	+ paint bucket fill
	+ stamp to save frequently used objects
	+ palette of most recently used colors
	+ eyedropper to pickup a color
	+ straight line tool
	+ easy to add many objects to workspace as user desires
	+ object is automatically placed in workspace when drawing accepted
	- BUT undo history only stores one past action
	- BUT no block fill
Usability of Rule Toolkit	+ rule search (i.e. with a large variety of rules the construction kit has a search mechanism to search for rules)
	+ easy rule creation from construction kit with direct manipulation
	+ large selection of rules
	+ easy to add rules as object behaviors
	+ easy rule testing
	+ rule search (i.e. with a large variety of rules the construction kit has a search mechanism to search for rules)
	+ comprehensive rule creation toolkit
	+ random rule firing
	+ each object can fire at its own speed
	+ easy addition of sound clip to object
	+ creates a project that is cross platform independent
- BUT rule step test does not indicate which rule fires	
- BUT with many rules available the user may not remember where to find the rule or the exact name of the rule to use rule search	

In the area of learning and exploration, the important difference between the two systems is that SimBuilder has a wider variety of drawing tools, which makes creating

new objects very easy. SimBuilder also has a more comprehensive rule creation toolkit. With this system it is even possible to create objects and rules that use sounds as well as movement. Both environments have their own features to support running and exploring basic and interactive simulations. AgentSheets users must create their character's drawings using an single pixel based bit editor, whereas SimBuilder contains a more robust and typical drawing tool (e.g., similar to Microsoft Office drawing tools). For the feature "*adding objects to workspace*" we again see an advantage for SimBuilder: the two step process to add agents to a worksheet in AgentSheets, to select the object and a second click to select the drawing pencil tool and a third click to add it to the worksheet will be frustrating to users, in SimBuilder the user does not have perform the "*add object to workspace*" task for objects are automatically added as the last function of the drawing task. With respect to rule creation AgentSheets users must go through six steps for basic rule creation or eight steps if they need to add a conditional or method; the analogous task is three steps for a basic rule in SimBuilder and four steps if you need to add a conditional or method. In object behavior AgentSheets has the positive claim of "user can easily add behaviors", but SimBuilder has the additional positive claims of more comprehensive rule creation toolkit and rules that handle random behavior to give more realistic depictions, so in all four categories we have good evidence in support of the SimBuilder system.

6.2.2 Reuse Scenario and Claims

As for the learning scenarios, we contrasted the same reuse task as it might be conducted in both AgentSheets and SimBuilder. Again we begin with a shared task context, then instantiate the context for each of the systems. The background context for the reuse scenarios is illustrated in Table 6.9 and the detailed user interaction scenario describing AgentSheets reuse activity is illustrated in Table 6.10.

Table 6.9

*Task Context for Reuse Scenario**2) Teacher reuses food cycle model and adapts it to create ocean food cycle.*

A teacher wants to build a model of an ocean food cycle for her 5th grade class. The model would include underwater vegetation, which grows as result of receiving sunlight. The model contains fish that swim throughout it. Small fish survive by eating the vegetation and larger fish survive by eating the smaller fish. Fish that don't eat will die, and fish that eat will grow. Also if pollution is introduced into the model it will cause some fish and vegetation to become sick and die.

To begin the reuse session the teacher searches for a similar simulation that she can reuse to build a modified ocean food cycle. When she finds a suitable model fish world, she runs the simulation to find out what the model does. She identifies agents and behaviors to access, which will be useful to her modified simulation. She remembers how to create new agents from her recent learning tutorial and begins to add necessary agents that don't exist in this model. For example, she creates a new seaweed agent). She remembers that she has seen an agent the Sun that caused the flowers to grow. She would like to reuse that functionality and decides to copy and reuse the Sun, whose Sunrays will cause the Seaweed to grow. She would also like to reuse the behavior of flower growing in the Flower Garden Model to give her seaweed a life cycle. She keeps the basic fish agents and decides to create a new agent, Whale that will eat the Shark, which currently eats Goldfish. She does this by getting a Whale from clip art or by copying Shark and making it larger. She also is able to reuse the behavior of Shark by analogy. The Whale eats a Shark, just like a Shark eats a Goldfish.

Details of the associated claims are as follows for AgentSheets (Table 6.11). The user will generally have a hard time reusing the example because only one world can be active at a time in AgentSheets. This may have been intentional on the part of the designer to keep the novice user from getting confused with too many windows. However, this makes reuse harder; if the user could open more worlds they would be better able to learn by example.

Table 6.10

User interaction scenario describing AgentSheets reuse activity

2a) *Teacher reuses food cycle, adapts it to create ocean food cycle w/AgentSheets.*

To begin the reuse session the teacher searches for a similar simulation that she can reuse to build a modified ocean food cycle. In each model she will have to open the project and open a worksheet to see the simulation. When she finds a suitable model fish world, she runs the simulation to find out what the model does. She identifies agents and behaviors to access, which will be useful to her modified simulation. She will press "edit behavior" to see the rules for an agent. She remembers how to create new agents from her recent learning tutorial and begins to add necessary agents that don't exist in this model. For example, she creates a new seaweed agent, presses "new agent", then inputs the predetermined size for the agent (i.e. 16x16 or 32x32 pixels), next step is to enter the name of the agent, then the system creates a template agent, the user must clear this AgentSheets logo from the template. She will then use the single pixel based drawing tool to create seaweed).

She remembers that she has seen the Sun agent before in the water cycle model and decides to copy and reuse the Sun, whose Sunrays will cause the Seaweed to grow, but she can only open one model at a time and has to create the Sun based on her memory of what it did in the previous model (or the user could save this ocean food cycle model that they are working on, open up the second model to refresh their memory, and then return to this model).

She would also like to reuse the behavior of flower growing in the Flower Garden Model to give her seaweed a life cycle. She keeps the basic fish agents, but decides to create a new fish agent, Whale that will eat the Shark, which currently eats Goldfish. She draws a Whale or copies Shark and would like to make it larger, but is restricted by standard icon size. From the copy of shark she wants to modify the behavior to eat sharks, but the behavior can't be modified, so the user will just stick with drawing a new Whale. The user creates a new Whale following the same process as they did in the creation of the seaweed agent. Then the user will add rules to the whale to swim and eat sharks.

Currently, there are only two methods of reuse in AgentSheets. One is cloning of an agent, which is an exact copy of the agent's look and behavior. Another way is to import an agent through the Behavior exchange. The system supports multiple looks or depictions, therefore the user may modify the agent's look, but unfortunately the user cannot adapt the behavior of the clone. Studies have been conducted with an AgentSheets' analogy editor (Perrone & Repenning, 1998). This mechanism allows the user to reuse behavior by analogy (e.g. a car runs on the road, like a train runs on a track). Another claim that is very useful is that AgentSheets supports rules for classes. For

example, all classes of fish can have the same behaviors. However, this does require that all fish have the same exact behaviors.

Table 6.11

Claims Analysis of a Reuse Scenario: AgentSheets

Usage Feature	Possible Upsides (+) or Downsides (-) of Feature
Allowing only one world to be active at the same time	<ul style="list-style-type: none"> + prevents the user from getting confused with too many windows - BUT makes reuse harder, the user can reuse other agents and simulations from the behavior exchange, but not within the AS environment
Supporting the renaming of agents	<ul style="list-style-type: none"> + enables a copy (clone) approach to adaptation and reuse of agents - BUT users cannot adapt clones behavior it is directly inherited from the base object that was cloned
Multiple depictions	<ul style="list-style-type: none"> + allows the user to have several related agents of the same class or type - BUT the appearance is the only thing that can be modified each new instance will share or directly inherit the same rule set
Analogy editor	<ul style="list-style-type: none"> + allows the user to reuse behavior analogy (e.g. a plant absorbs sunlight like a bigger fish eats a smaller fish)
Rules for classes	<ul style="list-style-type: none"> + all classes of fish could have the same behaviors - BUT this would restrict the fish to having the exact same behaviors

Again shifting our attention to the SimBuilder version of the reuse scenario, we elaborate the task goal as it might be pursued in the new environment (Table 6.12).

Table 6.12

User Interaction Scenario Describing SimBuilder Reuse Activity

2b) *Teacher reuses food cycle, adapts it to create ocean food cycle w/AgentSheets.*

To begin the reuse session the teacher searches for a similar simulation that she can reuse to build a modified ocean food cycle. The user has two available methods for opening simulations. The user can use the menu option "open a project from file" or user can return to the previous project or main interface where they can access all projects by clicking a project. When she finds a suitable model fish world, she runs the simulation to find out what the model does.

She identifies objects and behaviors to access, which will be useful to her modified simulation. She will select the object invoking its halo and "open a viewer" to see rules for an agent. She remembers how to create new objects from her recent learning tutorial and begins to add necessary objects that don't exist in this model. For example, she creates a new seaweed agent, selects the paintbrush and uses the nine point drawing toolkit to create seaweed. She would also like to reuse the behavior of flower growing in the Flower Garden.

She remembers that she has seen the flower object before in the water cycle model and decides to copy and reuse the Flower. She also wants to reuse Sunrays. The Sunrays caused the flower to grow and she want to use it in this model to cause the Seaweed to grow. She returns to the "previous model" copies the Flower and returns to the project with "next project" and pastes the Flower that can be reused or modified in this new simulation. She will repeat the same actions for Sun and Sunrays. Her model will have life with just the reuse of behaviors and the creation of a few new ones. She keeps the basic fish objects, but decides to create a new fish agent, Whale that will eat the Shark, which currently eats Goldfish. She draws a Whale or copies Shark and just selects its halo and "resizes it to make it bigger". From the copy of shark she wants to modify the behavior to eat sharks, and she modifies the behavior to delete the shark instead of the goldfish.

Details of the associated claims are the following for SimBuilder. The user is able to have as many worlds open as they can manage, which is helpful for a "copy-paste" style of reuse. However, the user may find it confusing if too many windows are open at once, because objects can be reused and shared by many playfields. SimBuilder supports the renaming of objects by just selecting the current name of the object and retyping it either with the object's halo or within the rule window. This can be very helpful when reusing characters that have similar look or behavior. For example, the system supports multiple depictions, supports easy changes between depictions with a cursor, and best of all, the new object's state can be modified. This means that each new instance can have distinct values of its internal variables, so internal state for a given character type need not be treated as a global variable.

Table 6.13

Claims Analysis of a Reuse Scenario: SimBuilder

Usage Feature	Possible Upsides (+) or Downsides (-) of Feature
Allowing multiple worlds to be active at the same time	<ul style="list-style-type: none"> + simplifies copy and edit of prior characters in building a new world + allows user to reuse any character and easily change it + allows user to copy behaviors from one world to another
Supporting the renaming of objects	<ul style="list-style-type: none"> + enables user to just select name and change it + enables user to change name in rule window
Multiple depictions	<ul style="list-style-type: none"> + allows the user to have several related objects + allows user to create animations easily with cursor tool + user can easily change between multiple depictions with mouse selection + appearance can be modified each new instance and each new instance can have a distinct variable state - BUT each instance that is created spawns a new process eating up system resources and slowing the system
Rules for classes	<ul style="list-style-type: none"> + all classes of objects could have the same behaviors

An important difference between the two systems is that AgentSheets uses the production-system model of condition-action pairs, where testing of conditions happens in an ordered fashion. With this model rules are interpreted and fired in the order in which they appear in the code. This makes the illustration of non-deterministic events very difficult to simulate (e.g., random behavior like a cloud moving). This model introduces many subtle complexities, and is perhaps the most difficult thing to debug. With SimBuilder we mitigate some of these complexities with an object-oriented system model, where rules are fired when and as requested rather than in a pre-determined order. Each object can decide what rule is fired in response to a request and at what speed its rules are stepped through. In other related systems we have observed that the computational engine may allow several levels of speed in a simulation, but none of them

support separately-controlled stepping speeds for each object. With the SimBuilder system we are more easily able to create simulations with non-deterministic events, which are enabled by a random number generator and individual execution speeds for objects. Both environments provide some debugging facilities (e.g., the ability to test all or part of a behavior by firing each rule individually), and a system step tool for stepwise execution of the entire simulation with SimBuilder, or the ability to test all or part of a behavior on a particular character with AgentSheets.

6.2.3 Implications and Conclusions

We performed analytic scenario and claims analyses of AgentSheets and SimBuilder with the goal of documenting the nature of the design improvements made in the new environment. To some extent this can be seen as a “validity check”, in other words a test of whether our environment supported the guidelines suggested in the requirements analysis. Our analysis thus far paints a positive picture. At an intermediate level, AgentSheets is robust in that it offers conditions/actions templates, and has an environment very conducive for exploration and reuse at various levels, but the SimBuilder environment offers more rule templates and is also conducive for exploration. It is also more convenient for novice reuse strategies such as copying and pasting from one project to another, because multiple simulations can be open and in view simultaneously. For special-purpose modeling, both SimBuilder and AgentSheets give the user the flexibility of extending rule palettes. These palettes are a customizable part of the environment. An experienced user of the system can create a new primitive to extend the existing set.

We also documented usability issues for the drawing toolkit. For example, we anticipate that an improved drawing facility within SimBuilder will make it easier to use. The new tool offers a wider variety of drawing, fill, copying, erasing tools, color selection tools and stamps that offer considerably more functionality and flexibility than those provided in AgentSheets. The drawing facility in AgentSheets is limited to a pixel sized drawing tool, eraser and block fill tool. The two environments also vary in ease of

use of drawing features and user interaction techniques. An error that persisted from user to user in the AgentSheets environment was just adding new agents to a worksheet. This error was caused by the dual functionality of the pencil tool, which was used to both draw the object and to reselect it to add the agent to the environment. In SimBuilder there was no analogous error for when the user completes drawing an object it is automatically added to the workspace. Both environments support user interaction with drawing tools through direct manipulation techniques. As a summary of drawing claims 9 of 15 were positive claims for AgentSheets and 12 of 14 were positive claims for SimBuilder, indicating that more positive claims favored the SimBuilder environment, therefore the drawing toolkit scenario and claims analysis supports the drawing tools and techniques of SimBuilder.

The programming style supporting novice teacher visual programming incorporated in both environments is a mix of textual and iconic language. Limiting a system to direct manipulation and icons can cause a computational complexity and rigidity enforced by the implicit nature of graphical rewrite rules. The general limitations implicit within graphical rewrite rules and their scalability to more complex problems were discussed by Kirsch: “It is not clear, however, how to extend the implicit underlying model used here to other pictorial sources of greater interest and importance.” Graphical rules as defined and utilized in AgentSheets can be confusing to users and a system that utilizes only visual look and position of an object as a trigger for firing a rule may be limited. Our solution is an object-oriented system that can react to other agents with message passing, which will help generalize solutions to reduce some confusion of during programming and creating simulations. As a summary of claims for the visual programming toolkit, we found 5 of 8 claims were positive for AgentSheets and 11 of 13 claims were positive for SimBuilder, therefore the scenario and claims analysis of programming style of visual programming toolkit favors SimBuilder.

As for the informal usability inspection, the “results” of the claims analysis should be seen simply as consistent with our design goals—many of the features and consequences that we have called out in this section reflect the limitations identified in

AgentSheets and related tools during requirements analysis and our subsequent efforts to address these limitations in the design of SimBuilder. However by analyzing the two environments at this level—in particular by thinking through the actual interaction scenario—we were able to document a number of issues and features hypothesized to distinguish the two. Thus these analytic studies were important in guiding the development of tasks and materials for the empirical study comparing learning and reuse in AgentSheets versus SimBuilder. The next section will illustrate in detail the AgentSheets and SimBuilder comparative study.

6.3 Empirical Comparison of AgentSheets and SimBuilder

In earlier work (Rosson & Seals, 2001; Seals, 2002) we studied the learning and reuse behaviors of novices working with AgentSheets to build visual simulations. This work helped to form the foundation on which SimBuilder was built, as well as providing experimental materials and instruments that were useful in teaching and assessing programming behavior. Thus after the SimBuilder environment had been constructed, we designed a similar study with the goal of comparing SimBuilder and AgentSheets. The major goal was to determine if SimBuilder produced the predicted performance and satisfaction benefits; a secondary experimental goal was to determine whether novices more easily reused abstract simulation components than concrete simulation components.

In the following sections we first describe the general experimental methods and procedures, including the dependent measures. We then report the results obtained, including both quantitative and qualitative data. We conclude with a discussion of how the results obtained relate to the research questions and hypotheses posed in Chapter 3.

6.3.1 Experimental Methods

This section discusses the general methodological concerns for the empirical study conducted in Phase III, the comparative evaluation. These concerns included study population and sample descriptions, apparatus, and experiment design.

6.3.1.1 Population and Sample

The target population of this research is K-12 teachers, in particular middle school science teachers and pre-service teachers. We have worked with many science teachers in this area, but with a small population of teachers available in the population, we chose to focus on pre-service teachers as participants, because as university researchers we have more convenient access to teachers who are still preparing for their careers. Each of our participants was self-selected from a graduate class in educational applications of technology at Virginia Tech. All the participants have classroom experience—they either have taught in the past, or currently teach in the classroom. The AgentSheets study included twelve subjects and the SimBuilder study included seven subjects (this was caused by self selection of students with fewer students volunteering in the class recruited as participants for the SimBuilder trial); all were compensated with a small stipend of \$15 for a session of 1.5-2 hours. Their participation was also an extra credit activity, but our goal was to get these pre-service teachers interested and trained in the use of our programs, with the hope that they will continue to use these educational software programs as they begin their teaching careers.

6.3.1.2 Apparatus: Hardware and Software

The study was conducted in the Virginia Tech Human Computer Interaction Usability Laboratory that was comprised of an experiment room and an evaluation room connected with a two-way mirror. The evaluator was located in a separate room to reduce user distraction. In episodes of serious breakdown, the evaluator was able to communicate with the user via an intercom. Video, audio and screen activity were recorded, and critical incidents observed were noted. The evaluator is able to see the participant while they are completing their activities, but the evaluator is obscured from the participant's sight.

In the experiment room, testing was conducted on a Gateway 2000e CPU with a 17" Sony Trinitron Multiscan 17seII Monitor running Windows 2000 Professional equipped with a standard scroll mouse. Other equipment consisted of the following: an Azden two channel WM Pro Wireless Transmitter with mini wireless microphone to

record participant discourse and facilitate think aloud study, two Gateway speakers. Also there is a Panasonic VHS-C Camcorder (Palmcorder) to record all the user body language and reactions. The Palmcorder is atop a tripod equipped with a Power Panner to adjust the camera remotely from the evaluation studio. A Scan Do Ultra scan converter was used to capture all screen events. The hardware used in the evaluation studio included a Laptop computer with Observational Coding Software for critical incident logging and VHS tape time stripping capability. A professional editing Hi-Fi Stereo S-VHS Video Cassette Recorder was used to record all experimental data, and a Videonics Digital Video Mixer was used to coordinate the view of video from user screen, video of user and audio of discourse. We utilized speakers and headphones to hear the participants and a two-way RadioShack intercom to talk to the participant if any serious breakdowns occurred (see Figure 6.2 for evaluator studio equipment details).

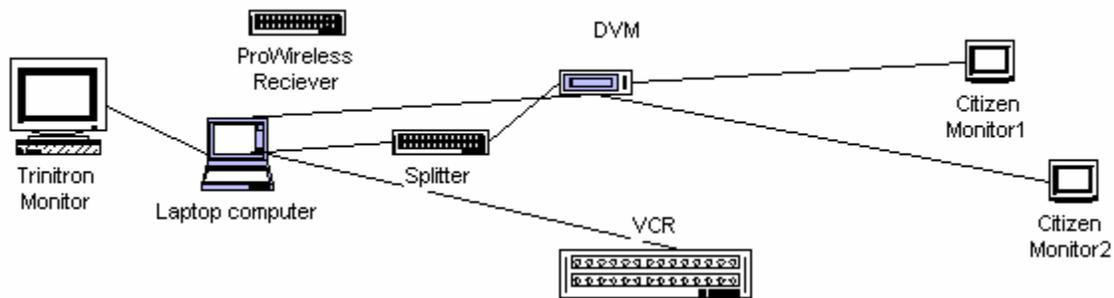


FIGURE 6.2. EVALUATOR APPARATUS SETUP

6.3.1.3 Experimental Design

The experimental design for the comparative study included one between-subjects factor (AgentSheets vs. SimBuilder) and one within-subjects factor (tutorial phase). The between-subjects conditions were administered at different times due to availability of the SimBuilder tool, but the data were collected using the same procedures and thus have been combined for analysis. Because the tutorial activities within the Learning phase (explore water cycle vs. create a volcano) are not comparable with those of the Reuse phase (reuse components vs. reuse an example), the Learning and Reuse portions of the experiment were considered to be separate designs rather than an extended set of repeated measures. A schematic of the resulting design can be seen in Tables 6.14 and 6.15.

Table 6.14

Design of Learning Experiment (Within-subjects: Learning phase)

Between-subjects: Environment	AgentSheets S1...S12	Explore water cycle	Create volcano
	SimBuilder S13...S19	Explore water cycle	Create volcano

Table 6.15

Design of Reuse Experiment (Within-subjects: Reuse phase)

Between-subjects: Environment	AgentSheets S1...S12	Reuse ozone as example	Reuse components from starter world
	SimBuilder S13...S19	Reuse ozone as example	Reuse components from starter world

Because one of our experimental questions concerned the reuse of different sorts of materials, we took care in the design not to confound the project given to users as a reuse task with the form of the reusable materials; similarly it was important to control for the order in which the conditions were administered, to ward against practice effects. A partially-counterbalanced design was used to control for these two factors. There were two reuse projects (ocean world and photosynthesis world) and two reuse materials (the ozone example and the components in starter world). This produced four possible combinations of reuse project with reuse materials; complementary pairs were assigned to different subjects, varying the order in each case. Because of the relatively small number of subjects a full counterbalancing was not possible, but we were able to achieve the following distribution of reuse projects, materials, and orderings across system type:

- AgentSheets: ocean-ozone followed by photosynthesis-starter (3 subjects)
- photosynthesis-starter followed by ocean-ozone (3 subjects)
- ocean-starter followed by photosynthesis-ozone (3 subjects)
- photosynthesis-ozone followed by ocean-starter (3 subjects)
- SimBuilder: ocean-ozone followed by photosynthesis-starter (2 subjects)
- photosynthesis-starter followed by ocean-ozone (2 subjects)
- ocean-starter followed by photosynthesis-ozone (2 subjects)
- photosynthesis-ozone followed by ocean-starter (1 subjects)

The design summarized in Tables 6.14 and 6.15 allowed us to carry out several sorts of analyses relevant to the research questions. First and foremost, it was possible to ask whether SimBuilder users enjoyed a general advantage over AgentSheets users, in both the Learning and the Reuse phases of the experiment. Secondly, it allowed us to determine whether reuse through concrete examples was more advantageous than reuse through abstract components.

6.3.2 Materials

The materials for this include the informed consent that users signed before participating in the experiment and the tutorial materials prepared to guide users through their learning and reuse sessions.

6.3.2.1 Informed Consent

The Institutional Review Board of Virginia Tech requires researchers conducting any research involving the investigation of individual characteristics or behavior involving surveys, interviews, or human factors evaluation to have board approval of research designs. The informed consent form states in written form for the participant of the purpose of the study, justification, procedures, benefits, and risks of the project, as well as guarantees the participant that all of responses are confidential and are used only anonymously (see Appendix D for IRB approval and Appendix E for Informed Consent).

6.3.2.2 Minimalist Tutorials

The participants were trained using minimalist instructional techniques (Carroll, 1990). The minimalist strategy that we utilize in this experiment is that of guided exploration cards. To support learning through exploration, we crafted this set of cards to train the users in the basic use of the environment, how to build simulations in this environment, and how to reuse simulations with task and open-ended exercises. Each guided exploration card was created as a presentation slide that contains a task and can be broken into several subtasks. Two sets of tutorial materials were created, one for the learning session and one for the reuse session. Two versions of each tutorial were developed, one for AgentSheets, and one for SimBuilder, with extreme care taken to

ensure that the tasks, level of support, and style of instruction was as similar as possible. In the end, there was a page by page mapping between the two tutorials.

The tutorials were derived from a similar tutorial created to study learning and reuse of AgentSheets in an earlier experiment (Rosson & Seals, 2001, see Chapter 4). As in this earlier work, efforts were taken to minimize the amount of verbiage, reading, and passive observation expected of students, and provide many open-ended questions to stimulate discussion and exploration. As additional support for users in efforts to avoid or recover from errors, images of the proper ending state for many programming steps were provided, and checkpoint explanations and tips for error recovery were inserted at key points (Anderson, 1993; Carroll, 1996; Rosson, 2002). In general, the supports for minimalism that were incorporated into the tutorial included the following:

- Exploration of complex simulation as a first activity to increase meaningfulness.
- Support for rapid start up by guiding learners to immediately begin augmenting and creating the code of this simulation by investigating objects, and their encapsulated state
- Visual cues (i.e. snapshots) of manipulatives to be utilized during their task in efforts to reduce errors
- Example simulations to help users learn by analogy and learners expected to reuse through recall and/or copy-paste reuse techniques

The tutorial for the learning session was 10 pages long and included an interaction guide to give users helpful hints to remind users of frequently used tasks. It begins with the user exploring a water cycle simulation, including several questions that lead them to discover specific features of the water cycle. Next the user explores an individual object in the simulation — in Figure 6.3 we see an example from the AgentSheets tutorial and Figure 6.4 we see the same example from the SimBuilder tutorial. With these guided exploration the learner is instructed to change the behavior of the Cloud object used by the water cycle simulation. The tasks were to change the rate at which the Cloud moved and to change the direction in which the cloud moved. By emphasizing example-based

learning we hope the user can learn to analyze or create behavior for an object by inspection and modeling. The complete tutorials are found in Appendix G and H.

Changing The Behavior of Clouds

 *Take a moment to review the interaction guide to gain a better understanding of the interactions between agent's behaviors*

Make Clouds more active. Currently the behavior of the cloud is to move 2% of the time. Find the IF statement that causes this behavior. Let's make the clouds move a lot more.

- Click on the desired text box (currently showing 2%) to make your change.
- **Increase the percentage to 50%** for moving to the right.
- Click  (located on the menu at the bottom of the behavior window).
- Now,  the simulation to see the affects of your changes.
- Press  and  so that you can try another change.

Make the clouds move vertically. Currently the behavior of the cloud is to move horizontally across the sky. Let's try to change the behavior of the cloud so that it will move vertically. Look at the rule to **move** agents.



- In order to change the direction that the agent moves in just click on the arrow and move it until it points down.
-  your new change.
-  the simulation to see how the simulation has changed.

AgentSheets Tutorial © Virginia Tech Visual Languages Group

FIGURE 6.3. AGENTSHEETS LEARNING SESSION: CHANGING BEHAVIOR OF CLOUDS TUTORIAL PAGE

Changing The Behavior of Clouds

- Examine some of the more complicated behaviors by selecting the *Cloud* and reviewing its behavior.

Take a moment to review the interaction guide to gain a better understanding of the interactions between player's behaviors

Make Clouds more active.
Currently the behavior of the cloud is to move forward 5 spaces. Find the **script** that causes this behavior. Let's make the clouds move a lot more.

- Select the desired script **cloud moving** and drag it to a clear workspace.
- Increase the value for moving forward to **10**.
 - Press **go** to see how your changes affect the simulation.
 - Press **stop** and try another change.

Make the clouds move vertically.
Currently the behavior of the cloud is to move horizontally across the sky. Let's try to change the behavior of the cloud so that it will move vertically.

- <Alt> Click** or **<Middle button> Click** your cloud and its *Halo* will appear.
- Select **Rotate** and move your cloud just a tiny bit for its direction arrow to appear.
- In order to change the direction that the *player* moves Click on the **green arrow** and change its direction so that it points to the **down**.
- Press **go** to see how the simulation has changed.

Squeak SimBuilder Tutorial © Virginia Tech Visual Languages Group

FIGURE 6.4. SIMBUILDER LEARNING SESSION: CHANGING BEHAVIOR OF CLOUDS TUTORIAL PAGE

After the learner has explored the water cycle microworld, answered a few questions about how it works, and added a new object of their own creation (e.g. a flying bird), he or she begins part two of the learning tutorial, a performance task designed to both practice (for the learner) and assess (for the experimenter) what has been learned thus far. The user is asked to create a volcano model using what they have learned by exploring and modifying the water cycle (see Figure 6.5). In creating the volcano simulation the user is given no object or rule creation clues, and provides the learners an opportunity to show their understanding or level of mastery of the environment by creating a tangible artifact.

Creating a Volcano Simulation

- To Leave WaterCycle
Select Navigator and Press PREVIOUS project.
- Now that you are back in the Welcome page.
Select Navigator and Press NEW project at the end of Navigator.
- Click Unnamed1 at the bottom of the new window and Replace it with *volcano_yourinitials*.
- Click  to begin a New Project.
- Once you have an idea of the new environment you want to create, begin by creating *new players*. You can use a your  to paint whatever you like.
- You can create a *mountain*, *sparks (that fly out of the volcano)*, *lava* and any other players that will improve the aesthetic view of your playground. Perhaps you would like to include a sky for background, or trees, etc. If you need help drawing a player, refer to the interaction guide.

Squeak SimBuilder Tutorial © Virginia Tech Visual Languages Group-Draft

FIGURE 6.5. SIMBUILDER LEARNING SESSION: CREATING A VOLCANO TUTORIAL PAGE

The tutorials created to guide the second (reuse) session consisted of 9 pages. Here the learner’s task is to create two simulations by reusing either an example-based or component-based reuse model. Recall that in the reuse sessions, one experimental objective was to compare the usefulness of a concrete example world versus a set of generic components as reusable materials. Thus we were careful to ensure that the two types of reusable models offered the same functionality; they contain objects that exhibit the exact same general behavior, but differ in their visual appearance. The “example-based” model was designed to convey a concrete example simulation, while the “component-based” model was designed to convey a collection of abstract elements.

More specifically, the example-based model is a simulation that could be used in an earth or general science classroom; it is an ozone depletion simulation. In this model a factory creates pollution, the pollution moves through the air until it comes in contact with the sun object that causes a chemical change to the pollution that breaks it down into

another element bromine and chloride BrCl, which if they come into contact with the ozone layer, the BrCl will deplete the ozone layer and replace it with weaker ozone. The user would read the tutorial page, run the model, and investigate each object. Finally to reinforce the users high-level understanding of each object, a detailed description of the objects actions is provided for the AgentSheets example-based model (see Figure 6.6).

In contrast, the component-based model is does not simulate a real world situation with interacting physical elements. Instead, its elements are generic components; we named it “starter world” to convey even more that these are reusable components. There is an emitter that instantiates moving objects (i.e. movers), the movers move though the simulation unless they come in contact with the changer object that causes the object to change into a random mover object. The random mover object moves around until it comes into contact with a replacer object, at which point it is replaced by an object that changes, when it is contacted by the replacer object. The generic model may seem complicated when described in text, but its elements provide the same operation object-object interaction as the ozone depletion model. We provided both of these models so that we could study the phenomena of reuse based on realistic examples vs. generic components. The user would read the tutorial page, run the model, and investigate each object. Finally to reinforce the users high-level understanding of each object, a detailed description of the objects actions is also provided for the SimBuilder component-based model (see Figure 6.7).

Reusing Ozone World

In the Ozone depletion Cycle a factory emits CFC into the atmosphere and a heterogeneous reaction takes place. This reaction converts the inactive chlorine and bromine reservoirs to more active form. No ozone loss occurs until sunlight initiates the catalytic ozone destruction.

- Open the Ozone Depletion Simulation.
- Now,  this model.
 - *Investigate each agent to discover its' behavior.*
- The Smoke_stack agent emits chemicals into the atmosphere.
-  The Chemicals are emitted by the smoke stack and move up into the atmosphere. They are changed into active BrCl when contacted by the sun.
-  The Sun agent replaces the inactive chemicals with active BrCl.
-  The BrCl agent moves randomly until it contacts an ozone agent.
-  The ozone absorbs (erases) BrCl and is changed into a weaker ozone agent
- Press  after a few minutes of observing the model.
-  Refer to interactions guide for *Help*

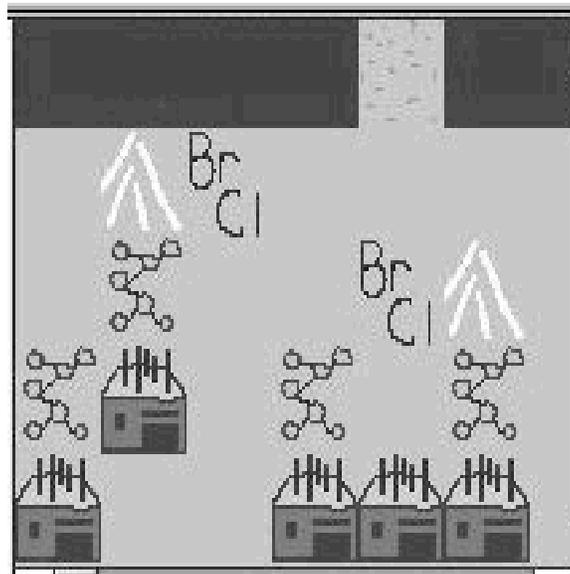


FIGURE 6.6. EXAMPLE-BASED REUSE IN AGENTSHEETS TUTORIAL PAGE AND OZONE SIMULATION

Reusing the Starter World

The Starter World is a simulation that contains some of the basic functionality of SimBuilder programs. It contains players that move, erase, emit other players, and change themselves into other forms.

- Open the Starter World Simulation.
 - Now, **Press  to start this model.**
 - *Investigate each player to discover its' behavior.*
-  The Mover just moves in one direction.
 -  The Emitter produces another player.
 -  The Eraser erases other players that it contacts.
 -  The Replacer replaces the Mover with another player.
 -  The Changer will change another player into a new player when it comes in contact with it.
- Press  after a few minutes of observing the model.
-  Refer to interactions guide for *Help*.

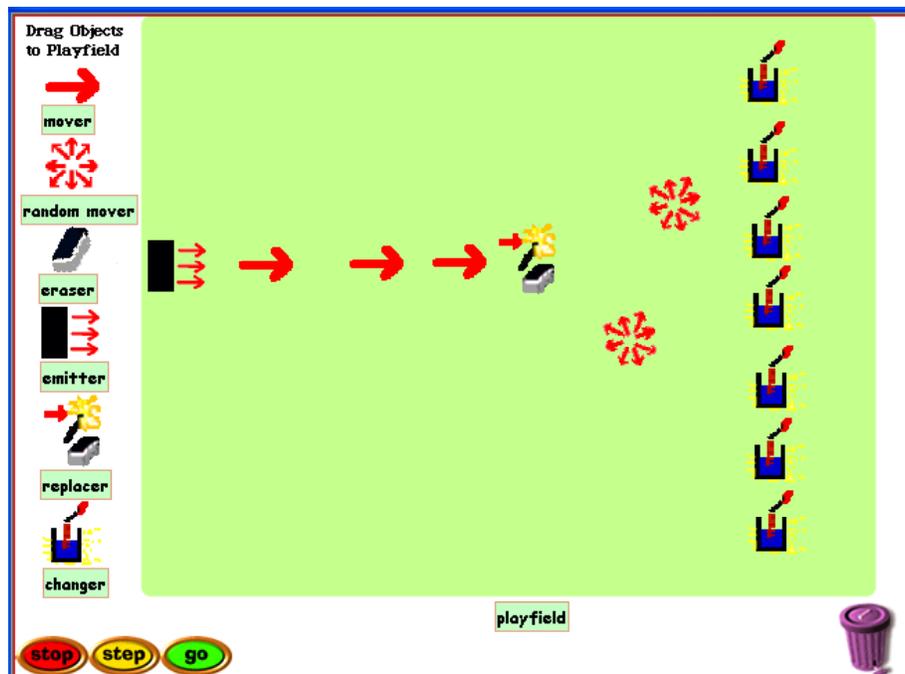


FIGURE 6.7. COMPONENT-BASED REUSE IN SIMBUILDER TUTORIAL PAGE AND STARTER SIMULATION

Creating new players and Adding behaviors to create Photosynthesis Simulation.

A simulation in SimBuilder is simply a set of players that work together to create visual effects.

A key aspect of reusing a simulation is to reuse existing players and add new players. We will begin with the small task of reusing a new *player* from the Ozone World.

Task 1. Create a new project
 In the **Navigator** and Press **NEW** project.
 ▪ Click **Unnamed1** at the bottom of the new window and
 ▪ Rename it **PhotosynthesisYourInitials**
 Click the **Photosynthesis** Project to enter it.

Task 2. Reusing and creating new behavior for your new player
 ▪ Some players already have behavior scripts. You may need to look at their behaviors to get started. You can also reuse a player by just using the grey *repaint brush* to change their look and just reuse its behavior

Task 3.  **new players.**  Refer to interactions guide if you need *Help*.
 ▪ Add behaviors for new players you create.  Refer to interactions guide for *Help*.

Think of other interactions to make your Photosynthesis simulation interesting.

- Press **PUBLISH IT!** and then **Save on local disk only**
- Press **< PREV** and return to the Welcome page.



FIGURE 6.8. SIMBUILDER MINIMALIST TUTORIAL PAGE: REUSE SESSION

After studying the provided reusable simulation model the user is instructed to create a new simulation (see Figure 6.8). The teachers will carry out this activity twice and create both a photosynthesis model and an ocean model (see Table 6.16 for a synopsis of tutorial activities). As described in the Experimental Design, the combination and ordering of reusable material and reuse project was determined by counterbalancing. Thus four versions of the reuse tutorial were created for each system version, where one version directed the user to create the photosynthesis model using the ozone example world, followed by the ocean model using the generic components, and so on.

Table 6.16

Summary of Learning and Reuse Tutorial Activities

Learning Tutorial	Reuse Tutorial
<ul style="list-style-type: none"> • Watch, answer questions about water cycle simulation 	<ul style="list-style-type: none"> • Watch, explore, answer questions about first model to be reused
<ul style="list-style-type: none"> • Explore and modify behaviors of Cloud in the water cycle 	<ul style="list-style-type: none"> • Identify objects in model to be reused that are candidates for project
<ul style="list-style-type: none"> • Add a bird object to water cycle 	<ul style="list-style-type: none"> • Create a new simulation reusing the candidate objects if possible
<ul style="list-style-type: none"> • Based on training create a volcano simulation 	<ul style="list-style-type: none"> • Watch, explore, answer questions about second model to be reused
<ul style="list-style-type: none"> • Test volcano simulation 	<ul style="list-style-type: none"> • Identify objects in second model to be reused that are candidates for project
	<ul style="list-style-type: none"> • Create a second simulation reusing the candidate objects if possible

As a final resource for the learners, a one-page interaction guide was created for each system to aid in error recovery and as a review of screen icons and helpful shortcuts. The interaction guide contains a pictorial listing of all the major dialogs, buttons, icons, and menus that the user interacts with during their experiment as an aid to boost short term memory and is mainly supplied just as a reminder of how to perform tasks that the user has been introduced to during their experiment. Figure 6.6 shows the AgentSheets Interaction Guide.

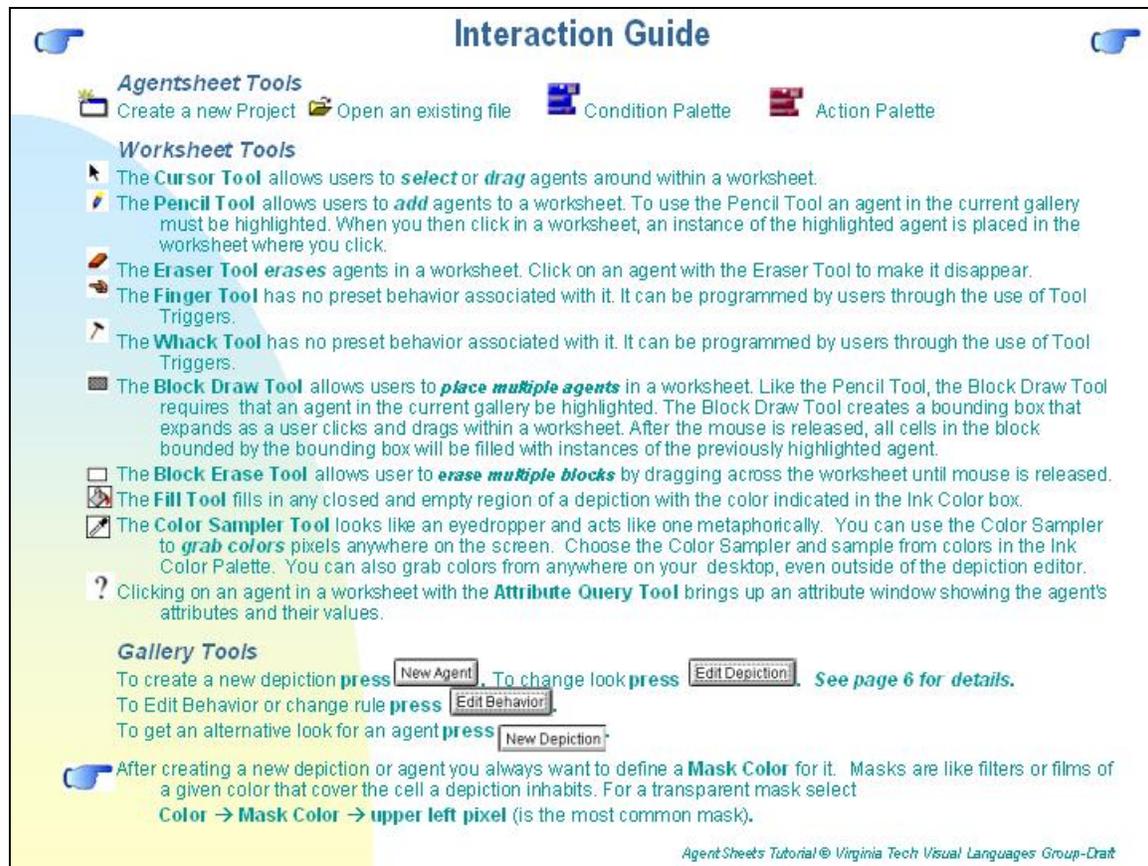


FIGURE 6.9. INTERACTION GUIDE

6.3.3 Procedures

The experiment began by having users read both a university institutional review board approval for the experiment and an informed consent form. This material familiarized them with the experiment and allowed them the opportunity either to sign the informed consent and begin the experiment or to decline to participate. Next, users completed an on-line user background questionnaire that was utilized to determine both their baseline comfort level with computers and whether they met the criteria established to determine user suitability for participation in the experiment. After users completed the background questions, they were moved to the evaluation suite and given the experiment instructions. The evaluator instructed them in the use of the think aloud protocol, explained that their experience was to be guided by their tutorial, and informed them that they could take a brief break between the two sessions involved in the testing. The

evaluator also gave them an orientation to the experimental computer setup, and reminded them of the details of their informed consent (e.g., audio/video would be recorded, but that their responses would not be attributed to an individual (i.e., they were guaranteed anonymity) unless they gave express written consent to have their name attached to the data they provided).

Users participated in two sessions (a learning session and a reuse session with a 5-10 minute break in-between sessions). Users' introduction to the environment culminated at the end of the learning session in their creating a Volcano model. This demonstrated their mastery of the environment by the creation of a tangible artifact. In the reuse session, users created two simulations with two types of reuse – generic component-based reuse and example-based reuse. These created simulations were a photosynthesis model and an ocean model. Some participants used the ozone example as reusable material for photosynthesis, others used the components of starter world for photosynthesis, some did the photosynthesis project first, others doing it second, and so on, all according to the counterbalancing model described earlier (Section 6.3.1.3).

Our intent was that the participants independently perform the experiment with little intervention from the evaluator, relying only on the minimalist instructional manual. However, in some cases the experimenter was forced to intervene, following these guidelines:

1. Tell the participant to reread the section,
2. Tell the participant to reference the Interaction Guide,
3. In case of severe breakdown, actually help the user to recover.

Once users finished the experiment sessions, they retired from the experiment suite and were repositioned at a data collection machine to complete a Post-test Questionnaire, utilized to gather subjective reactions to the two environments. Finally, they completed a retrospective interview to allow the evaluator to collect any last participant thoughts.

6.3.4 Experimental Data Collection

Table 6.17 provides an overview of the experimental instruments and measures. Each user completed a detailed questionnaire before the session; the questions assessed whether the user met the entrance criteria for the experiment, compiled user background, demographics, and their expectations about simulation creation. During and after the experiment we gathered performance data consisting of session timing, numbers of objects created, number of rules created, complexity of rules created, and simulations users created during performance tests (see Appendices L & M). We videotaped the sessions and recorded detailed observations and comments during the session. At the conclusion of the experiment users left the experiment studio and completed an on-line post-test questionnaire on a separate data collection machine. After the questionnaires were complete the participants were given a retrospective interview for debriefing and elaboration. The final piece of experimental data was an expert evaluation of the participants' work products.

Table 6.17

Experimental Instruments and Measures

Instrument	Description
Pre-test Questionnaire	User background, demographics, expectations, etc.
Performance data	Time, types and # of rules and objects, user created simulations as performance artifacts
User Observations	Qualitative observations and critical incidents
Post-test Questionnaire	User satisfaction and system ratings
Retrospective Interviews	Debriefing and elaboration
Expert Evaluation	UI experts' ratings of teacher simulation

6.3.4.1 Pre-test Questionnaire

The pre-experiment questionnaire gathered general information about the participants to assess whether they met the criteria established for classification as both novice programmers and content area experts. Data gathered included such general

identifiers as age, gender, and major, as well as data about their educational background and their experience levels in software design, computer programming, curricula, and instruction. It also included questions about their professional background to gauge whether work experience or teaching experience would affect their performance.

The second group of questions ascertained participants' familiarity with computers. It posed questions about how long they had used a computer, as well as certain computer applications, drawing software, word processors, spreadsheet programs, chat, etc. The third group posed questions about users' (teachers') views on the use of simulations in the classroom – how they believed simulations work and what kinds of simulations they could imagine creating. These questions were designed to provide a pre-post test comparison. These questions also provided data on how much users had increased their knowledge of either visual programming or simulations and on whether their exposure to simulation manipulation had increased their ability to conceptualize a wider range of simulation applications. The fourth section was a series of Likert scales designed to gauge participants' previous exposure to general computing terminology (see Appendix F for details of Pre-test Questionnaire).

6.3.4.2 Performance Data and User Observations

Performance data was collected in the form of user-created artifacts (i.e., exercise results) during the guided exploration provided by the minimalist tutorials. Working from these simulation artifacts, we were able to assess the time to create artifacts as well as the number of components or objects created within each artifact. We were also able to investigate the raw numbers of rules the user created and the complexity of the rules created (i.e. whether the rules were basic, intermediate or advanced).

In addition to the performance data, we collected informal user observations as well as more formal observations in the form of critical incidents: Fitts and Jones initially codified critical incidents in 1947 in their analysis of pilot error in reading and interpreting aircraft instruments. Critical incidents have been further refined from a software perspective to mean an interaction with an element of the system or a feature that is either particularly easy or difficult for the user and, as a result, affects performance

by causing either extremely good or extremely poor performance (del Galdo et al., 1987). The critical incident technique was proposed in 1954 by Flanagan and studied by Castillo (1997) and Hartson and associates (1996). When evaluators use this technique, they identify and report critical incidents, both negative and positive, and consider how these critical incidents affected user performance.

6.3.4.3 Post-test Questionnaire

The post-experiment questionnaire was designed to gather detailed information about how participants assessed their performance and the systems used, as well as to judge on a qualitative level whether—and if so to what extent—users had learned more about the visual educational simulation domain. Details of the post-test questionnaire can be found in Appendix I.

Part one of the questionnaire gathered overall user reactions to the system using six bi-polar rating scales. Participants rated their general satisfaction with the simulation environment (i.e., SimBuilder or AgentSheets, depending on their experimental condition) according to six contrasting scales: Terrible/Wonderful, Frustrating/Satisfying, Dull/Stimulating, Difficult/Easy, Rigid/Flexible, and Boring/Interesting. The second part of the questionnaire included a series of 28 Likert-type scales, where participants rated the strength of their agreement to statements concerning whether they found the system to be easy to use, easy to get started using, fun for building simulations, and so on. The third part of the questionnaire assessed participants' understanding of the simulation projects they had just completed, as well as their ability to specify requirements for other educational simulations that they might use or build in the future (for details see Appendix J).

6.3.4.4 Retrospective Interviews

The retrospective interview format was designed to capture any information or statements users provided either as critiques or affirmations of the environment's success. Interviews consisted of six questions, which asked users to share their feelings about the

exercise, as well as any suggestions they had for improving the experience and any other simulation ideas they might have (for questions see Appendix K).

6.3.4.5 User Interface Expert Evaluation

As a converging measure of teachers' success in using SimBuilder versus AgentSheets, we developed a procedure whereby experts in human-computer interaction could evaluate the visual and behavioral aesthetics of the teacher' work products. We developed a form to initiate and guide the rating process. The first section of the evaluation was developed to evaluate and validate user interface expert certification. A number of criteria were used to indicate whether evaluators had sufficient experience in the areas of software engineering and design; and usability engineering, design, and interface evaluation. The second section of the evaluation introduced evaluators to the tasks and informed them that the evaluation being conducted was to ascertain the aesthetic quality of these artifacts by using two individual ratings, one for rating the actions of characters, and one for their visual representation. The third section consisted of a rating guideline review (Shneiderman, 1998) designed to provide consistency. It gave evaluators concrete examples of anticipated ratings. The rating guideline (see Table 6.18 explicitly gave users well-defined criteria for assessing whether a simulation expressed the intended real world situation. The final section contained the simulations to be rated, as well as a summary of the embedded characters' actions.

Table 6.18.

Guidelines for Expert Ratings of Participant Simulations**Center for HCI: User Interface Evaluation Rating Guide**

Ratings	Action Representation	
5	Excellent	A rating of Excellent indicates that the user models the phenomena with a variety of relevant working objects (5 or more).
4	Above Average	A rating of Above Average indicates the user has fairly realistic depiction with at least 4 working relevant objects.
3	Average	A rating of Average indicates the user has a realistic depiction with at least 2-3 objects working.
2	Below Average	A rating of Below Average indicates the user has a at least one object working.
1	Poor	A rating of Poor indicates that the user model has no working objects.

Ratings	Visual Representation	
5	Excellent	A rating of Excellent indicates that the user had good realistic drawing that models the phenomena with a good use of color and 5 or more objects.
4	Above Average	A rating of Above Average indicates the user has fair use of color and realistic depiction with at least 4 relevant objects.
3	Average	A rating of Average indicates the user has fair use of color and realistic depiction with at least 3 relevant objects.
2	Below Average	A rating of Below Average indicates the user has a fair use of color and realism with at least 2-3 objects.
1	Poor	A rating of Poor indicates that the user vaguely conveyed the model with 2-4 objects.

6.3.5 Results and Analysis

In this section we present the results from the empirical comparison of SimBuilder and AgentSheets, including both quantitative and qualitative data and analysis. We begin with a summary of participant background obtained from the pre-test questionnaires. We follow this with analyses of quantitative and qualitative data collected in the learning sessions and in the reuse sessions respectively, in each case raising and considering the evidence with respect to the experimental hypotheses. In a separate section, we compare participants' reactions to the two systems and finally conclude with a discussion of expert ratings of the simulations produced in the two environments.

6.3.5.1 Participant Background

For practical reasons data for the two experimental conditions (AgentSheets and SimBuilder) were collected at different times. However, the questions from the demographic and screening questionnaire allow us to compare the characteristics of each group. A summary comparison of several quantitative measures appears in Table 6.19.

Table 6.19.

Participant Background Data

	AgentSheets (N=12)	SimBuilder (N=7)
Average age	27	26.5
Percent female	75%	72%
Percent with teaching experience	58%	66%
Average years of computer use	12	14

We circulated a sign up sheet in the two classes to recruit participants, therefore all were self-selected. We initially had 18 participants volunteer from the class that worked with AgentSheets and 13 participants to volunteer from the class that worked with SimBuilder. Our initial experiment design was to have 16 in both trial and with every four participants getting the exact same treatment. During experimentation with scheduling conflicts, class conflicts and no shows our final experiment consisted of 12 and 7 participants in the AgentSheets and SimBuilder trial.

The ages of the 12 AgentSheets participants ranged from 22 to 45 with a mean age of 27 years; twenty-five percent of these participants were male and seventy-five percent were female (this gender ratio is consistent with the majority of K-12 schoolteachers being female). The declared majors of these pre-service teachers were very similar, including Education Curriculum and Instruction, Elementary Education, and English Education. With respect to software experience, none of the participants had been exposed to software design, but all had previously done some web page design and used Macromedia Dreamweaver MX, Netscape Composer, or Microsoft Front Page; one AgentSheets participant had taken an educational technology course. All but one of these participants were currently enrolled in their first Instructional Technology class. The group had not been exposed to computer programming; only one participant had taken an undergraduate programming class, and this was more than four years earlier (she could not even remember what language she had used). The majority of these participants had taken at least one computer technology class. None of the participants had any work experience that would be useful in software design or programming. In answer to a question about their comfort using computers, half of the group (six participants) rated

themselves as “good with computers”. In general, this group felt that they were computer literate and comfortable enough to install new software on their computers.

Fifty eight percent of the AgentSheets participants had teaching experience. All of them had considerable experience with computers, with a mean of 12 years experience (the range was 5-16 years). Eighty-three percent of the teachers have used both Macintosh and PC computers, but most reported that they use PCs with more regularity. The majority have used computer games and drawing software (e.g. Adobe Photoshop, Microsoft Paint, Printshop, AutoCAD, etc.) All participants used computers in their classes, the Internet, email, and saw a role for computers in the classroom with responses like the following:

- Good for cause/effect relationships & to integrate technology into the curriculum
- Use in science experiments
- I see computers as a supplement to class activities, and to enrich the lessons.
- Technology is becoming more of a factor in today's society. With the use of technology in most aspects of life it is important for students to become familiar and comfortable with it. The use of computers and technology is making it easier to set up scenarios for students to learn.
- Provides students with real life situations they may otherwise not be able to experience.

When asked what kind of simulations they would build, the AgentSheets participants proposed the following: high & low tides; events of 9-11-2001; microevolution, plant or animal growth/development, metabolic processes, an earthquake, a walk through nature; virtual tours of places; battle/wars, volcanoes erupting, destruction from hurricane or earthquake; growth of cities; building collapses, tidal waves, earthquakes, meteor impacts etc; court hearings, dramatizations of plays, novels, and other literary works, historical moments; conflict management, how to put on a program, publicizing programs/event; hurricanes.

As summarized in Table 6.19, the seven SimBuilder participants had a similar background profile. Their ages ranged from 20 to 32 with a mean age of 26.5 years; in this case 72% of them were female. Like the AgentSheets group, the participants' majors were similar and included Education Curriculum and Instruction, Higher Education, and so on. As for the AgentSheets group, none of these participants had been exposed to software design, although all had previously done some web page design and used Dreamweaver MX, Netscape Composer, or Microsoft Front Page; again, one had taken an educational technology course. All of the participants except one were enrolled in their first Instructional Technology class. The group had not been exposed to computer programming – as for the group, only one participant had taken an undergraduate programming class over 4 years prior and again could not remember what language had been used. None of the participants had any work experience that would be useful in software design or programming. With respect to comfort using computers, the entire group rated themselves as “good with computers”. Like the AgentSheets group, these pre-service teachers felt that they were computer literate and felt comfortable enough to install new software on their computers.

Slightly more of the SimBuilder participants had actual teaching experience relative to the group — 66% versus the 58% reported for the first group. The average years of experience using computers was 14 years, with a range from 5 – 16 years. 83% of the teachers had used both Mac and PC, but again most use PCs with more regularity. The majority have used computer games and used drawing software (e.g. Adobe Photoshop, Microsoft Paint, Printshop, AutoCAD, etc.) All had used computers in their classes, the Internet, email, and saw a role for computers in the classroom with responses like the following:

- I see computer simulations playing a huge role in primary/secondary education because the advancement of technology has grown so much that it allows for such simulations to be created. Also, the use of visual aides has grown increasingly as students discover that they help to learn material better. Use in science experiments
- I think it would be useful for students to learn in a different way.

- Computer simulations can offer students a visual way of learning/processing valuable information and at the same time train them to be adept at computer usage before entering higher education facilities.
- Computers help to enhance the student's educational experience - assisting student's evolution into independent learners
- Visual simulations tools will help children see the entire processes front start to finish
- As instructional tool for students

When asked what kind of simulations they would build this group proposed a set quite similar to those gathered from the AgentSheets group: simulations of storms and how they are created, simulation of historical battles, exploration of cities through computer simulation; Something with outer space; Car accident, fire alarm, tornado alert; Earth Quakes; traffic accidents, getting lost, moving into a new home, searching for/buying a new car.

The background questionnaires show that both groups were composed of individuals who were reasonable representatives of our target population — although not all had real world teaching experience, all were currently enrolled in teacher education programs. However, the questionnaires did suggest that the seven pre-service teachers making up the SimBuilder group might have had more computer experience than those who were in the group. To determine if this was the case, we conducted an analysis of participants' reported experience with computing technology to determine whether the groups were indeed equivalent on this factor.

To create a measure of general computer experience, we first examined the reliability of the Computer User Experience scale (CUE), a well-tested scale for assessing self-reported computer expertise (Potosky & Bobko 1998). The high internal reliability (Cronbach alpha = 0.88) suggested that this was an appropriate metric of participants' computing background. The average normalized scale value (z-score) for the AgentSheets group was -0.39 (SD = .62) and for the SimBuilder group was 0.66 (SD =

.43). An independent samples t-test of the two groups confirmed that the SimBuilder user group self-rated themselves as higher in computer experience than the AgentSheets participant group and was statistically significant [$F(1,18) = 15.59, p < .001$]. Consequently the subsequent between-subjects comparisons will consider in each case whether CUE scores should be included as a covariate.

6.3.5.2 Learning Sessions: Performance Data

The first phase of the learning session included participants' guided exploration and modification of the water cycle model. It was followed by a second phase in which they worked primarily on their own to design and create a simulation of a volcano. Because of the ordering of these two phases and the relative degree of help in the first versus the second phase, we refer to these as the "Training" and the "Performance" phases. Table 6.20 shows the average times (in minutes) for participants completing these two phases of the learning session, as well as their entire learning session times. The training time was measured from the time that the user began reading their tutorial until they completed the last activity in the Water Cycle Exploration; the performance time was measured from the time that the user began reading the information to create the volcano simulation until the end of this exercise.

Table 6.20.

Learning Session Times for SimBuilder versus AgentSheets Groups

	SimBuilder Average Minutes (SD) N=7+	AgentSheets Average Minutes (SD) N=11
Training (water cycle)*	25.29 (2.22)	35.06 (7.89)
Performance (volcano)	23.63 (17.34)	29.92 (10.86)
Total learning time	57.07 (17.66)	65.24 (15.22)

+The data from one AgentSheets learner was lost due to equipment malfunction.

* Significant difference at $p < .10$

As suggested by these mean values, there seems to be a tendency for the SimBuilder users to work more quickly through their learning activities than those using AgentSheets, particularly during the first phase of the tutorial. However there was also

considerable variability among individual users, and an uneven number of users in each group. The observation that the standard deviations were considerably increased for the volcano project relative to the water cycle exploration suggested that these are rather different activities, and indeed bivariate correlations of training and performance times confirmed that there was no significant relation among these 18 participants [$r(17) = .29$, ns]. However, analyses of computer experience and these performance measures revealed that training time was related to CUE scores [$r(18) = .53$, $p < .05$]; performance time was unrelated to computer experience [$r(18) = .29$, ns]. Throughout this work we will indicate significance at $\alpha \leq 0.10$. This liberal value was chosen because of the exploratory nature of the work, small sample size, and tremendous individual variability.

To determine whether the apparent mean differences were reliable, two MANOVA analyses (one for training, one for performance) were conducted using System as the single independent variable with two levels (AgentSheets and SimBuilder). The analysis of training times revealed a significant statistical difference [$F(1,16) = 9.99$, $p < .01$]; however the difference between performance means was not significant [$F(1,16) = 0.91$, ns]. When a significant difference was found with Computer User Experience, it is necessary to perform an ANCOVA to factor out the effects of this factor (Stevens, 1978). To determine if computer experience differences between the groups was driving the observed mean difference, an ANCOVA was conducted with CUE as the covariate, System as a between-subjects factor, and training time as the dependent measure. Including CUE covariate reduced the strength of the difference; despite a difference in the raw means, statistical analysis was not significant [$F(1,15) = 2.88$, $p < .12$].

An additional set of analyses focused on the *complexity of the artifacts* that the participants produced in their learning sessions, namely the volcano projects they designed and implemented in the performance phase. For each participant, the volcano project was decomposed with respect to the number of objects it contained, and the number of behavior-specifying rules it contained. A further distinction was made among rules created, classifying them as either basic (typically a simple movement like object forward by five); intermediate (an object that moves but also incorporates additional

functionality such as sound, or interaction with other objects); and expert (rules that used advanced concepts like local or global variables, function calls, triggers, and so on). In practice, so few rules were at the expert level that only frequencies for basic and intermediate rules were analyzed. Table 6.21 compares SimBuilder and AgentSheets learners with respect to the objects and rules created during the volcano project.

Table 6.21.

Artifact Complexity Measures for SimBuilder vs. Learning

	SimBuilder Average # of Objects or Rules (SD)	AgentSheets Average # of Objects or Rules (SD)
Volcano objects	4.85 (2.54)	4.63 (1.68)
Volcano rules	4.85 (4.84)	2.54 (1.54)
Volcano basic rules*	3.14 (2.47)	1.54 (1.21)
Volcano intermediate rules	1.71 (2.42)	0.90 (0.53)

* Significant difference at $p < .10$

As for the learning session times, the means suggest a general tendency for SimBuilder learners to perform at a higher level than their AgentSheets counterparts. In each case, the means for the SimBuilder group are larger than those of the AgentSheets group, although again the variability within subjects is large. A MANOVA analysis of these data indicated that the difference in basic rules was statistically significant [$F(1,16) = 3.39, p < .09$]. None of the other comparisons were significant.

In order to determine whether computer experience was a factor in these performance data, bivariate correlations of the four complexity measures and the normalized CUE scores were obtained. These analyses revealed that none of the complexity measures were correlated with computer experience, thus no further analyses were conducted.

Because of the large individual variability, the small and unequal sample size, and the differences between groups in computer experience, it is difficult to draw strong conclusions from the learning performance data. However, the combination of results are promising, suggesting that SimBuilder learners may have enjoyed both an initial learning

time advantage during the training phase (guided closely by the tutorials) and at least some degree of simulation complexity advantage during the performance phase (where users were essentially “on their own” to create the volcano). We turn now to a more qualitative discussion in support of these tentative conclusions.

6.3.5.3 Learning Sessions: Qualitative Observations

In general, the learners completed the tutorial activities with little intervention from the facilitator. The learning session began with a water cycle simulation. The learners had little trouble running and exploring this example simulation, although one of the SimBuilder users had trouble finding the object “halo” used to access the object’s features and behavior. This particular problem seemed to be due to issues of mapping mouse keys to selection tasks (using the middle button of a 3-button mouse).

Learners were guided to create a new object to add to the water cycle simulation, and the comparative flexibility of the drawing toolkits became apparent during this task: users complained that the drawing tools provided by AgentSheets were too rigid. In contrast, SimBuilder users seemed to be more satisfied with the drawing tools and created more realistic depictions as characters. For example, compare Figure 6.10 (created with AgentSheets) and Figure 6.11 (created in SimBuilder). Both depict volcanic activity, but the second is a more expressive and visually realistic rendition of this, using arrows, rocks, and other details to show how the pressure builds, the explosion occurs, and other life forms (birds, etc) flee the chaos.



FIGURE 6.10. AGENTSHEETS VOLCANO

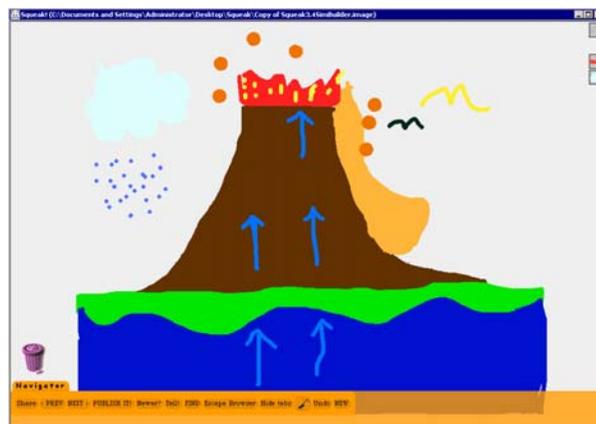


FIGURE 6.11. SIMBUILDER VOLCANO

During both the water cycle and volcano activities, we observed that learners working with SimBuilder benefited from its drawing tool's flexible point size, undo, flexible object size and paint fill. For example, we observed that the SimBuilder's use of a "crosshair at the tip of the paint fill bucket" was more successful than the analogous AgentSheets "tip of the drop of paint connected to paint fill bucket". In AgentSheets most users assumed that the center mass of the paint bucket was where the fill would occur and this caused many fill errors to occur (most users had errors in using this bucket fill and filled the entire area and had to repeat their creation or undo the paint error). AgentSheets users also complained about the size of their creations (AgentSheets characters are either 32x32 pixels or 64x64 pixels) and wanted an option to make some objects different sizes to emphasize scale.

Teachers working with AgentSheets started out quickly by investigating the water cycle model, but had many errors initially with selecting objects and drawing. Once users had created and saved a new object, they had problems adding that new object to the workspace—they were confused by the fact that they must use the same user interface control to both create an instance of the object and select it for addition to the workspace. User also had problems viewing the behaviors of objects. The user had to open a window to view an object's rules, but needed at least 2-3 extra windows to create rules. Users also had problems changing the direction of objects and getting basic rules to work. All of the users' rules worked semantically, but in the system did not work syntactically because of extra conditions that the users needed to negotiate.

During the learning sessions with SimBuilder, the participants also started out quickly by investigating the water cycle model. However in general they did not experience problems in the selection of objects and use of drawing tools. They were able to begin creating an object with one selection of the paintbrush and once they finished their creation it was automatically added to the workspace. Users could easily duplicate objects and add multimedia such as sound to their objects. User also had little problems viewing the behaviors of objects. Once the user invoked the user halo or handle they

would open a viewer to see an object's rules. When creating additional rules the user would have to drag a rule from the viewer or rule palette. Users had problems initially changing the direction of objects; it consisted of a two-step process of invoking the handle and changing the direction pointer. Users had no problem getting basic rules to work by just dragging them from the rule palette.

6.3.5.4 Reuse Sessions: Performance Data

Because of the counterbalanced nature of the reuse tasks (i.e., the order in which they were attempted and the reusable material provided), the performance data for the reuse sessions can be summarized in two different ways: we can look at the two reuse projects simply as two tasks ordered in time (this mixes together the two different types of reusable material), or we can analyze them with respect to reuse type (this mixes together the order in which the tasks were attempted). With respect to practice, one would expect the second reuse project to have an advantage regardless of reuse material provided; however our experimental hypothesis of interest concerned a predicted advantage of generic components over concrete examples. As a result, we have provided means and standard deviations of the reuse session times using both of these possible data organizations (Table 6.22).

Table 6.22.

Reuse Session Times for SimBuilder vs. AgentSheets

	SimBuilder Average Minutes (SD) N=5*	Average AgentSheets Minutes (SD) N=11
Reuse project 1	20.60 (10.54)	24.21 (4.71)
Reuse project 2	21.49 (13.62)	22.96 (5.12)
Reuse concrete example	20.50 (11.52)	23.91 (4.77)
Reuse generic components	20.47 (12.34)	22.62 (5.06)

* The reuse timing data from two SimBuilder users was lost due to equipment malfunction.

A quick examination of the mean differences suggests that the differences if any are slight. Although there is again a trend for the AgentSheets users to “take longer” to

carry out the reuse sessions than the SimBuilder users, these differences are very small in comparison to the individual variability among subjects (The generally-larger variability for the SimBuilder group can be attributed at least partially to the significantly smaller group size). Repeated measures ANOVAS (using Reuse1 vs. Reuse2 and Examples vs. Components as within-subjects variables) confirmed that there were no significant differences among the two systems, the two reuse sessions (i.e., 1 vs. 2), or the reuse materials (i.e. example vs. component).

As for the learning sessions, we also calculated several measures related to the complexity of the simulations produced by the users during their reuse projects—in this case the objects and rules created for both the project reusing the ozone example and the project reusing the starter components. In this case, because our interest was primarily on the effect of reuse material, we have summarized the number of objects and rules created, as well as the number of basic and intermediate rules, broken down by system used and type of reusable material. Table 6.23 shows the mean values for each measure.

Table 6.23.

Artifact Complexity Measures for SimBuilder vs. AgentSheets Reuse

	SimBuilder Reuse Simulations (SD) N=7	AgentSheets Reuse Simulations (SD) N=12	Combined across Systems (SD) N=18
Example: objects	3.86 (0.38)	3.50 (0.85)	3.65 (0.70)
Example: rules*	3.29 (0.76)	2.10 (0.99)	2.59 (1.06)
Example: basic rules	2.14 (1.35)	1.30 (0.68)	1.65 (1.06)
Example: intermediate rules	1.14 (0.69)	0.80 (1.03)	0.94 (0.90)
Component: objects	3.43 (0.54)	3.30 (1.16)	3.35 (0.93)
Component: rules*	2.57 (0.54)	3.10 (1.79)	2.88 (1.41)
Component: basic rules	2.00 (0.82)	2.00 (1.49)	2.00 (1.23)
Component: intermediate rules	0.57 (0.54)	1.10 (0.88)	0.88 (0.78)

* Significant difference at $p < .01$

A review of the means suggests that the differences if any are modest—in only a few cases are the projects created with the ozone example as reusable material different than those created with the starter world. If anything, it appears that there may be a complexity advantage for projects created with the concrete example, just the reverse of what had been predicted. However, repeated-measures ANOVAs (with types of reuse material as a within-subjects variable and system as between-subjects) revealed that none of these differences were statistically reliable. Thus just as for the reuse session times, we cannot conclude anything about the relative benefits of concrete examples versus generic components as reusable materials.

Interestingly, these analyses revealed a significant interaction for the measure of simulation rules, where users working with SimBuilder created simulations with more rules when they were reusing the concrete ozone simulation, while AgentSheets users created simulations with more rules when reusing the generic starter simulation [$F(1,15) = 10.42, p < .01$]. A similar significant interaction was found for the measure of intermediate rules [$F(1,15) = 3.97, p < .10$]. This is interesting because our original prediction had been based on work done using the *AgentSheets system*, and in fact these results seem to replicate that earlier finding. However, the inverse relation for SimBuilder is very curious. We speculate that this may be related somehow to the motivational characteristics of SimBuilder versus AgentSheets examples. Perhaps SimBuilder simulations are more fun to look at and create, and working in the presence of another “interesting” example like ozone world motivated greater effort on the part of the SimBuilder users. An alternative explanation is simply that the reusable materials in the two worlds were not exactly equivalent and this may have influenced participants’ reaction to and reuse of them.

6.3.5.5 Reuse Sessions: Qualitative Observations

Immediately following the learning tasks, users started to work on the tasks that were designed to investigate their reuse of examples versus components. During these sessions, each reuse project began with study of the assigned reuse example—either the component-based simulation (see Figure 6.7 for an example of this simulation in

SimBuilder) or the example-based simulation (see Figure 6.6 for an example of this material developed in AgentSheets). Each of these simulations provides example representations and code for characters able to move, move randomly, emit other characters, transform another character and erase another character (Rosson & Seals, 2001).

The participants' reuse task was to first design a quick paper rendition of the target simulation (recall that this was either a photosynthesis model or an ocean world). After planning the simulation's basic objects and behaviors, the participants' next task was to explore the model simulation, either the ozone depletion model or the starter component model. After studying the model, participants were directed to assess if and how it offered objects that could be good candidates for reuse in creating their new simulations. They began with investigation and exploration of the model, followed by a set of new simulation initiation steps (e.g. creating a background). If they had identified a character that seemed to be a good reuse candidate, they would reuse that object either by recalling how it worked or using a copy-paste technique. Note however, that (intentionally) the two tools provided differential support for reuse of other characters. For example, in AgentSheets users were able to use example characters as models, or even change the look of an example character, but they could not modify or extend its behavior. In contrast, SimBuilder users could select a reusable object and make a copy or place that object into their new project and make any changes they desired (e.g. name, behaviors, depiction, etc.).

A good solution for the photosynthesis reuse project developed by an AgentSheets user is shown in Figure 6.12, where the user has created five objects. The action of the simulation is the following: the sun creates sunrays, which travel down and to the right. Once a sunray shines on or contacts a small tree, it grows or transforms into a large tree. A good SimBuilder solution for the ocean project is shown in Figure 6.13, a model also comprised of five objects. The actions of this ocean simulation include clouds, which float, an ocean which produces waves and a tropical island that the waves crash into.

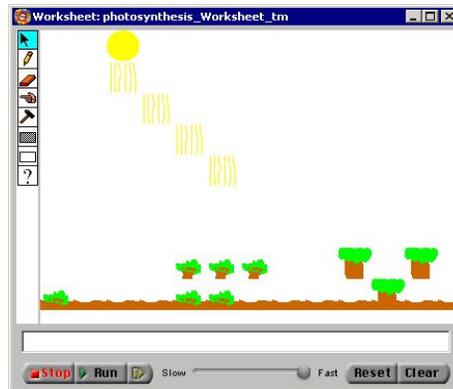


FIGURE 6.12. PHOTOSYNTHESIS MODEL



FIGURE 6.13. SIMBUILDER OCEAN WORLD MODEL

It was difficult to judge participants' reactions to the two types of reusable materials. During the phase of the session when they were guided to open and explore either the ozone world or the starter world, they generally simply started by opening the world. In response to the tutorial probe question of "can you find anything that would be useful to your project?", most users were able to identify something that seemed useful. But the specific comments were tied to the project that they were attempting, for example, a number of users in AgentSheets and Stagecast saw a connection between the ocean character and the emitter in the starter world. Apparently the semantic concept behind the emitter was similar enough to that of an ocean "producing" waves that the reuse opportunity was easy to spot. In contrast, the same opportunity for using emitter as a model for the sun (creating rays) in photosynthesis seemed to be more difficult to see, and users did not jump to it as readily.

In response to the probes during study of the ozone pollution simulation, the focus of users' reactions appeared to be the visual representation of the world. In this case the object providing the "emission" functionality (a factory that created chemical molecules) had a rather trivial visual difference from for example, the ocean: the factory emits chemicals in an upward direction, whereas waves need to be created in a sideways direction (as does the emitter object in starter world). So here we see that very detailed aspects of the reusable objects seemed to influence participants' understanding of the potential for reuse, or perhaps their evaluation of how easy it would be to do the adaptation for their project.

Interestingly, users who worked first with the generic starter world may have received a more general benefit from the reusable world. For example, one person immediately saw the relation of emitter to ocean, and mover to wave, when doing her first (ocean) reuse project. When encountering the factory in service of the photosynthesis project, she used this same term in describing her analysis of factory, commenting that the sun would be "emitting" sunrays. So in her case, she did not map her needs to the factory objects, but continued (perhaps) to use the concept of an emitter to plan the behavior of a sun object. She went on to create an entirely new object to model the sun, showing no literal reuse. Cases like this suggest that a useful role for the starter world is simply to make explicit several useful "patterns" for simulation objects. Subsequent reuse then may take place at a conceptual more than code-reuse level. This account also offers an explanation for why the performance results are mixed, namely that for people who first worked with starter, the benefit may have continued through the second project. If so, an analysis that looked only at participants' first project might provide a better comparison of the two reuse sources.

6.3.5.6 User Reactions

The post-test questionnaire assessed users' reactions in two sets of rating scales, as well as with a series of more open-ended questions. The first set of rating scales asked participants to provide a rating on a bi-polar scale regarding several qualities that might be expected to influence usability. The means and standard deviations for these ratings,

broken down by the tool used, are shown in Table 6.24. For each of these scales, a higher rating indicates a number closer to the “positive” side of the scale (e.g., Wonderful, Satisfying). Thus a quick review of the means suggests that there was a tendency for SimBuilder users to react more positively than AgentSheets users.

Table 6.24.

Bi-polar Rating Scales Assessing General User Satisfaction

Bi-polar Scale Anchors	SimBuilder Ratings Mean (SD) N=7	AgentSheets Ratings Mean (SD) N=12
Terrible — Wonderful	3.71 (0.77)	3.33 (0.77)
Frustrating — Satisfying	3.00 (0.95)	2.67 (0.77)
Dull — Stimulating	4.14 (1.29)	3.50 (1.16)
Difficult — Easy	2.71 (1.24)	3.00 (0.95)
Rigid — Flexible*	3.71 (1.21)	2.67 (1.15)
Boring—Fun	3.87 (1.11)	3.25 (0.96)

* Significant difference at $p < .10$

The mean differences were tested using a MANOVA; the test revealed that none of the differences apparent in the bi-polar ratings were significant, although the difference for ratings on Rigid-Flexible was significance ($p < 0.10$).

The second set of rating scales was more extensive and consisted of a series of items designed to assess reactions to specific aspects of the participants’ learning and reuse activities. These scales were developed as Likert-type items, with each scale consisting of an assertion (e.g., “<AgentSheets> was easy to learn and use>”) to which participants responded using a 5-point scale from 1 = Strongly Disagree to 5 = Strongly Agree. There were 21 of these scales, with subsets of them designed to focus primarily on general ease of use, participants’ motivation to do simulation programming, reactions to the programming that they did, and the two different sorts of reusable material. The items have been classified according to their primary focus (ease of use, etc.) and organized into four corresponding tables (Table 6.25 – 6.28). Most of the items were written with a positive context such that a “5” would be a most positive reaction; some

items written with a negative context are noted using italicized text, and the ratings for these items have been recoded to be consistent with the others. Thus higher values in the SimBuilder column indicate more positive reactions to this system.

Table 6.25.

Likert-Style Rating Scales Assessing General Ease of Use

Likert-type Scale Item	SimBuilder Ratings Mean (SD) N=7	AgentSheets Ratings Mean (SD) N=12
1) Easy to learn and use	3.00 (1.41)	3.33 (0.77)
2) Easy to get started	3.57 (1.13)	2.67 (0.77)
3) <i>Hard to remember tool location</i> (NOT)	3.00 (1.29)	3.50 (1.16)
4) Easy for novices	2.71 (1.25)	3.00 (0.95)
6) I understand how to use	4.00 (1.00)	3.25 (0.96)
12) <i>It was hard to recover from errors</i> (NOT)	3.29 (0.48)	3.50 (0.79)

* indicates significance $p < .10$

A quick examination of the items designed to assess general ease of use suggest that reactions are mixed: in four cases the mean rating for the AgentSheets users is higher, and in two the reverse is true. To assess the reliability of these raw differences, a simple MANOVA with System as the only between-subjects variable was conducted. This revealed that none of the differences were statistically significant. Thus we cannot conclude anything from the rated satisfaction concerning these learning-related items.

Table 6.26.

Likert-Style Rating Scales Assessing Motivation

Likert-type Scale Item	SimBuilder Ratings Mean (SD) N=7	AgentSheets Ratings Mean (SD) N=12
5) Fun for building simulations*	4.43 (0.53)	2.67 (1.15)
7) I created a working simulation*	4.57 (0.53)	3.75 (0.45)
19) I can have objects any size I want*	4.00 (0.00)	2.33 (1.15)
20) I am enthusiastic about creating sims*	3.86(1.06)	3.08 (0.73)

* indicates significance $p < .10$

A quick examination of the items designed to assess motivation point to a much more consistent picture than those directed at general ease of use: in all cases the mean rating for the SimBuilder users is higher than that of the AgentSheets users. To assess the reliability of these raw differences, a simple MANOVA with System as the only between-subjects variable was conducted. This revealed that four of these mean differences were statistically significant: Fun building [$F(1,17) = 8.10, p < .05$]; Created working simulation [$F(1,17) = 12.79, p < .01$]; Objects any size [$F(1,17) = 14.23, p < .01$] and Enthusiastic about creating [$F(1,17) = 3.27, p < .10$]. Thus it seems clear that users who worked with SimBuilder were more enthusiastic and motivated after the experience.

Table 6.27.

Likert-Style Rating Scales Assessing Programming Style Reactions

Likert-type Scale Item	SimBuilder Ratings Mean (SD) N=7	AgentSheets Ratings Mean (SD) N=12
8) <i>Creating simulations was complicated</i> (NOT)	3.14 (1.46)	3.17 (0.85)
9) <i>Drag and drop rules were complicated</i> (NOT)	3.86 (0.97)	3.42 (1.24)
10) <i>Rule ordering was confusing</i> (NOT)	3.43 (0.48)	2.58 (0.90)
11) My simulation works logically*	3.29 (1.38)	2.45 (0.90)
13) <i>Creating rules was confusing</i> (NOT)	3.49 (0.97)	2.50 (0.67)
14) Rule creation was simple and natural	3.29 (0.95)	3.55 (0.80)
21) Visual representation a good match	4.00 (0.57)	4.09 (0.30)

* indicates significance $p < .10$

An examination of the items designed to assess programming style are again somewhat mixed, with several modest differences in favor of AgentSheets and several larger differences pointing to an advantage for SimBuilder. As for the other items, a MANOVA with System as the only between-subjects variable was conducted to evaluate these means differences. This revealed that most of the differences were not statistically reliable, but that the positive difference for SimBuilder ratings with respect to the item “Simulation works logically” was statistically significance [$F(1,14) = 3.05, p < .10$]. This suggests that the differences among rule creation and programming were not large, but

that there may have been a small effect of how “logical” the computational model was for the SimBuilder users.

Table 6.28.

Likert-Style Rating Scales Contrasting Reuse Style Reactions

Likert-type Scale Item	Combined Ratings Mean (SD) N=19
15) I understand reuse of ozone sim	3.15 (0.95)
17) I understand reuse of starter simulation*	3.68 (0.82)
16) I understand behavior of ozone sim	3.36 (0.96)
18) I understand behavior of starter sim	3.50 (1.04)

* indicates significance $p < .10$

Note that in Table 6.28, the items have not been broken down by System because the focus of this research question was on reusable materials (example vs. component). The items were designed to make two contrasts; one focused generally on reuse (items 15 and 17 in the table) and one more specifically on the behavior of the reusable objects (items 16 and 18). The mean differences do suggest that there may be a small advantage for the starter world objects. A repeated-measures ANOVAs comparing item 15 vs. 17 revealed that understanding of starter components was judged to be significantly greater than that of ozone example objects [$F(1,18) = 8.82, p < .01$]. An analogous test of item 16 vs. 18 revealed that this contrast in means was not significant. It is difficult to know what to conclude from this, but it does seem that the starter world did a better job of conveying what it means to reuse material in simulation creation.

Following the two sets of rating scales, 15 open-ended questions were presented, so as to gather a more qualitative picture of participants’ reactions. In the following numbered paragraphs, we summarize key elements of responses of the AgentSheets versus SimBuilder participants:

1. *What was most interesting or fun?* AgentSheets users made comments that some of the most interesting and fun things for them to do dealt with drawing and

coloring items. The second set of things that they liked about the environment was creating objects, creating a scenario and watching it play out, seeing the final results and actually seeing what the model did. Another comment was that actually creating the simulation and having a real life problem/event and visually seeing the process. One teacher commented that they liked working on the photosynthesis model. SimBuilder users made comments that it was interesting and fun to create an object and create rules for a simulation. Another user commented that s/he enjoyed creating the Volcano and having the sparks come from the volcano: seeing the objects, creating a volcano, reusing the emitter to create their model.

2. *What was least interesting or fun?* AgentSheets users commented that some of the least interesting and fun things for them to do dealt with creating behaviors, object interaction, and just getting familiar with the techniques and procedures for creating simulations. Another least interesting thing was to add behaviors that fulfilled the user's design of his/her model. Another user commented that the experience was not very user friendly at first, but after more exposure the program was easier to use, especially with help from the evaluator. One user commented that it was hard to use the condition and action menus and that they seemed to be based on very specific situations. One user had problems with the drawing software and stated, "It was hard to actually draw what I wanted. The drawing part was a little rigid." Another user commented that it was frustrating that there was not a better mix of auditory and visual learning and that not being able to change the picture size was frustrating.

SimBuilder users made comments that it was least interesting and fun was to try and figure out how to apply the rules and behaviors that they needed to complete their design. Other users commented that it was not interesting to work with the rules, and that it was not interesting to change or create the ocean erosion simulation. However there were many fewer comments of "least interesting" activities made by SimBuilder users (even when we consider the fewer numbers of participants).

3. *Did you find the example simulations used in the tutorial effective? Why or why not?* The AgentSheets users mainly commented that the tutorial was effective for many reasons. Some of the positive remarks were as follows: I work well from examples, examples are helpful, demonstrating an activity is helpful; it was good to review when developing a new simulation; I could copy and paste similar behaviors from them to the worksheet I was creating; the simulations were effective because they demonstrate an activity that is easy to understand and learn from; they were effective in showing what it was capable of; yeah I did, once I slowed them down I was able to see how it worked; it gave me ideas as to how to create my own simulation; yes, gave you an idea or foundation for how to set up the simulation.

Some of the less positive remarks from AgentSheets included: The examples were a little helpful, but were confusing because what I wanted to do and what the example did was two different things; I had to first think about what I wanted to do then look a look at the example and see if it was similar -- most cases it was similar to some very little degree, which was why it was so confusing; yes but it was still confusing at first with some of the graphics and tool bars.

The SimBuilder users also commented that the tutorial was effective for many reasons. Some of the positive remarks were as follows: Yes, the example simulations gave me a guideline for creating my own simulation; Some of the simulations were effective because of the visual effects; Studies have shown that some students excel in their studies by using visual/spatial intelligence as identified by Howard Gardner, Ph.D.; Yes - I think students could learn a lot from creating scientific simulations; Yes... they were interesting; They provided good examples; and simply Yes. A less positive remark was "They were o.k., but they are not very explanatory."

4. *Did you find the instructions in the tutorial helpful? Why or why not?* The AgentSheets users commented in many ways about whether the tutorial was helpful or not so helpful. Some of the positive remarks were as follows: Yes, it cleared up some confusion, especially the graphical representations of the tools in the various boxes (e.g.,

behavior, new agent, edit depiction, etc.); Yes-straightforward; Yes, the directions did a good job walking me through the exercise, I am not good at discovering computer steps on my own; Yes, allowed you to become familiar with the program, how to use it, and the different features of the program; tutorial was necessary in order to create your own simulation; for the most part, then did help; sometimes, the instructions assumed I knew exactly where to go. So more details might be a good thing to have. For example, describing everything word for word.

Some of the less positive remarks were the following: Yes but it helped when the instructor also walked through some of the things as well; they appeared to need more detail, they did however give me a better understanding in the specific areas; yes, but the behaviors were complex enough that it took trial and error; sort of, but personal help was better; I found them to be helpful. The screen captures could have been more clear on the instructions. But I understand that is how it is sometimes; instructions were sometimes difficult to understand & follow. This could present challenges to people who do not have strong computer skills; Yes, but the behaviors were complex enough that it took trial and error. They were a little confusing but I thought the tools were also.

The SimBuilder users commented in many ways about whether the tutorial was helpful or not so helpful. Some of the positive remarks were as follows: Yes - I was able to navigate effectively; Yes, very straightforward. The less positive remarks were “Yes, however, it was difficult to figure out how to correct an error” and the negative remarks were “The instructions seemed too confusing.” and “Many details were missing from the tutorial.” These last comments may be responding to the relative lack of refinement in the SimBuilder (i.e. versus AgentSheets) tutorial; recall that the AgentSheets tutorial had been based on one tested and refined in earlier research, whereas the SimBuilder tutorial was created specially for this project.

5. What 1-2 things would you change if you were asked to revise the tutorial? All AgentSheets users, except one, offered comments about how to revise the tutorial. Some of the remarks were as follows: Show it more in a step-by-step layout; Size of graphics,

graphics no so blocky looking make it so I can create one picture & have everything interact; Make instructions easier to understand for those not computer literate, include more graphics, use larger fonts for ease in reading; 1. Make the size of agents changeable after they're created 2. Simplify the behaviors; Just refine the beginning directions; I think that there were too many activities to do. By the third one I was ready to go because they were so complicated. I was used to seeing things in real life view versus what I drew so at times it was hard to put things together; Change the menus to add behaviors to the object, make them more self explanatory and easier to use, less specific with directionality; Have the desktop close when it opens, make the tools better, Does it only have to be science? Maybe you can use another content as an example. It would be interesting to see how English is applied; If there could be a way to add sound to the tutorial that would be helpful; I would make the behavior boxes clearer, I felt they were very hard to use. The dragging of boxes confused me as well. However, by the end I felt more comfortable with using the program.

All SimBuilder users commented on how to revise the tutorial. Some of the remarks were as follows: I would place more instruction about the rules and how to use them; Simplify the steps; Make the tutorial more detailed-with step-by-step screen shots. Make the rules for getting things to less complicated; The ocean erosion was a tad bit confusing; I don't know that I'd change anything, but I did appreciate the helpful guidance from the facilitator. This helped when I was lost or confused.

6. Suppose you were going to build a computer simulation of a volcano exploding for earth science. What sorts of things do you think would be involved (i.e. what objects and what do they do)? This question was asked in both pre and post-experiment, so that we could assess whether participating in this experiment changed the users' ideas about creating simulations. If they didn't have any idea about what to do initially but presented a clear plan in their post-assessment we would deem the experiment successful in introducing the user with a new way of thinking about the details of educational simulations. Participants in both groups were able to provide a good level of detail in their post-test answers to this question.

AgentSheets: Details of a volcano simulation from AgentSheets users.

- Geology, population density (e.g., human, animal, etc.), forestry, oceanography, etc. --- stages of volcanic eruption, lava, scenery, etc.
- Magma -> shoot out, lava- flow down, have things melt & bubble
- Sketch simulation, what do volcano, steam, lava, heat, sparks, etc., do (roles)
- A non-moving mountain flowing, lava, rising magma, rising smoke, and flashing lightening.
- Mountain, Lava, rocks, ...explosion
- A volcano, lava, smoke, dark clouds in the sky. The lave would come out of the volcano, the sky would have the dark clouds around it, and smoke would come out of the volcano as well.
- Pressure would build up under volcano, volcano would erupt lava, lava would produce heat, lava would produce smoke
- Volcano, atmosphere, sun, ground, sky, lava, changes in the environment around the volcano like the color of the land.
- Mountain lava smoke sky
- Volcano, ash, lava, ground, and gray background. The volcano shaking and the lava coming out of it would be cool too.
- Need a background, then a mountain. The lava would be coming out of the volcano because of pressure. The lava would shoot up the volcano, erupt down the sides of the volcano and onto the ground. There would be sparks that also are emitted from the top of the volcano.
- I would include lava or sparks hitting the volcano and show the sky becoming darker .

SimBuilder: Details of a volcano simulation from SimBuilder users.

- I would have to use the paintbrush to create a volcano, sparks, and background scenery. The sparks would fly out of the volcano to simulate explosion.
- Volcano, lava, sparks of lava rock, black and grayish colored smoke
- The things that were in the tutorial were fine. I cannot think of any more.
- Mountain - explosion of rock and lava an ocean that surrounds the mountain upward arrows to stimulate building pressure
- Volcano, earth, smoke, fire, sparks, lava
- I'd build volcano, lava, and lava spark objects. The lava spark objects would rise out of the volcano

7. *As well as you can, please describe what you think is the best way to come up with projects? (What criteria would you emphasize?)* After being exposed to AgentSheets, participants offered the following ideas about creating projects: The best way to "come up with projects" is to tap into student's passions, to emphasis their natural talents and gifts.; Math games (simulations of distance traveled, estimation, etc to find projects, look at some lessons on the web or in teacher manuals; think through, visualize (sketch on paper), experiment, revise as developing; Brainstorming, looking @ current periodicals. I would emphasize relevance to content matter and periodicals; Through imagination, and interest; I would look at the most complicated areas of study and then create projects; Tying them to curriculum (ex. what is being studied in science) use the

simulations to aid visual and kinesthetic learners and to help explain difficult science processes; Emphasize systems that are talked about in the classroom, make sure that the project incorporates; Imagination; See what interests students, and good research. Then create a lesson/project that will appeal to the students. Something they can relate to multiple steps and agents; I think that the best way is to see what subjects are going to be taught and what information is going to be given out. From there brainstorm to see if there are projects that could help the students learn if they could see a visual of it.

The ideas that teachers came up with to create projects after being exposed to SimBuilder were as follows: I would emphasize the use of rules to create movement; The best way to come up with projects should involve instructions that will take no more than 10-15 minutes to figure out since class time will be usually around 50-60 minutes. Also, students tend to tune out if instructions are too long and tedious; Having people get in a group to brainstorm; Using real life occurrences. Historical events such as volcanoes, earth quakes etc.; take the course outline and curriculum and come up with examples for lectures; brainstorming activities that relate to real-world experiences.

8. *How would you use this type of software?* There were a wide range of responses that teacher came up with for using this type software after being exposed to AgentSheets. Some of the positive ideas are as follows: Reinforce concepts taught in class: to promote inquiry based learning; To help students create their own simulations for better understanding of the subject matter; I could use this type of software for cause and effect relationship presentations; Mainly for science processes; I could use it to describe an action like the growth of a city or the pollution of the environment for the class before a lesson, or I could have a class do research and come up with their own simulation as a final project; For simple simulations; I don't know how I could use this software for English. Maybe I could use it in describing a scene in a novel or play, or I can use it to teach grammar lessons. Actually, now that I think about it, this software can help me give students more visuals; To show students how something works. Sometimes it is very helpful for people to see something; I would use this to help students investigate weather conditions or cause and effect problems. Several less optimistic responses were:

In social studies it would be near impossible, for example, how would I represent the Columbian exchange (e.g., plant and animal migration) from 1492 to 2002?; Unsure at present time; Perhaps. If its limitations would not limit the expression or understanding of concepts.

There were a many good ideas that teacher brainstormed for using this type software after being exposed to SimBuilder. Some of the ideas are as follows: I would use this type of software to show my students how things occur in nature or how something functions; This is good for science class and maybe math class in order to make the "what happens next" step be visual; I would use it in a small classroom setting. I would also use it as a tutorial for other applications such as breast self exams etc.; I would use this software as a form of interaction and visual learning; This software is useful in an earth (geology) science class; in a science class; this SW would be good for teaching earth and biological sciences in grade school.

9. Were you able to reuse the rules of any other agent? If so did this aid you in the creation of new agents in your simulation? Yes or No? Please explain. There were both affirmative and negative responses about whether the teachers were able to reuse agents within AgentSheets. Some of the positive responses were as follows: Yes, by the 3rd agent I felt pretty comfortable; Yes, It was helpful to refer back to when developing new agents behaviors; Yes it was the most useful thing I learned how to do. It minimized trial and error with behaviors. Yes, I could reuse drawn objects; Yes, I reused the rules of the last simulation to help me with the photosynthesis simulation. I reused the emitter, and the replacer and the random acts; yes I was able to reuse rules, and it was nice to have because it was already done for me. Also, now I know how to make waves move; Yes, Yes It was helpful to reuse things already done. If I couldn't use it exactly as it was set up I was able to see how it was already set up and then tailor to my need; yes it saves time in having to redraw or direct object.

Some of the less positive responses were as follows: No, because I wasn't sure how to get to the other agents. Whenever I went to "File", I was stopped from accessing

"Open". Plus, I enjoyed creating the agents; No, I couldn't figure out how to incorporate them into the ideas that I had formed; no the eraser didn't work properly; I was able to think of how the rules from one would help me, but I did not reuse rules exactly. I always had to change them to fit what I was doing.

There were only affirmative responses about whether the teachers were able to reuse objects within SimBuilder and their comments were as follows: Yes, I was able to reuse rules and this helped make the creation of my simulation faster and less confusing; Yes. The emitter function proved useful. It saved time in trying to program an object to move; yes, this was an aid because I did not have to make up the rules for that object; yes; yes. I reused a lot in the emitter simulation. That was straightforward to follow; yes. yes. I could simply copy rules instead of having to figure out how to write new ones. This was a bit confusing, but for the most part, it worked.

It is interesting to note that across both AgentSheets and SimBuilder, only the characters from the starter world received special mention in these open-ended questions (e.g., the emitter seems to have been seen as particularly useful). None of the participants mentioned the characters within ozone world, though of course this does not prove that they were thinking only of the starter world components.

10. Which operation is easier? Creating new agents and functionality from scratch or to reuse agents and functionality. Please explain. In the AgentSheets group, there were arguments for both options. In favor of creating new agents from scratch the following comments were made: Creating from scratch.; Creating new -> you don't have to mess w/someone else's work; Creating new ones is more fun, I like it better; Creating new agent from scratch. This is because by looking at what the example it I would get confused at to what I was asked to do; Creating new agents was easier because they were self drawn based on your own ideas; Creating my agents, but reusing the rules.

In contrast, AgentSheets users in favor of reuse offered the following: Modify behavior? Much easier to reuse agents & modify behavior?; Reusing, for the reason listed

above. Drawing is fun. Behaviors were a little tedious. Reusing agents was better; Reuse-I had a better concept of what something was. There were two users that argued for both options: They both seemed to be easy once I learned how to work the simulation; both became equally easy as time progressed, at first I would say creating a new agent was easier.

Within the SimBuilder group, all but one argued in favor of reusing objects: I thought it was easier to reuse the agents because the rules were already set in place; Reusing agents and functionality is easier because it can save you a few steps and give you more time for creativity in your construction of a simulation; To reuse agents was better. It saved time as well as frustration; REUSE! The volcano example wasn't as straightforward b/c I didn't really understand how to create rules, but in the emitter simulation I was able to reuse existing rules. It saves a lot of time; reusing agents is definitely easier. I didn't want to have to think about how to develop new ones. This seemed a bit complicated. (Perhaps if I had better experience w/the SW, I'd be more confident.). The one user arguing in favor of creating from scratch was "creating new agents - because I do not have to figure out what the other person previously constructed and why."

11. Can you think of any steps that would have made the reuse activity more straightforward? Some teachers in the AgentSheets group offered steps to improve the reuse activity: Make it easier to reopen previous galleries to access agents; Make sure the ask activity I would do and the example are very similar so I could reuse rules (behaviors and actions).; I think that the thing that made the reuse activity straightforward was when I used it and practiced it; Maybe have a reuse link as an option in the gallery.

A few teachers in the SimBuilder group also made suggestions for improvement: Having that stated in the tutorial to look at the objects as you can use them by copying over them and retitling them to do what you want for the project you need; An explanation of the construction; Better names.

12. *Would you be willing to use such a tool for creating educational simulations for use in your classroom? Please explain.* Some teachers in the AgentSheets group stated these opinions: In social studies it would be nearly impossible, for example, how would I represent the Columbian exchange (e.g., plant and animal migration) from 1492 to 2002?; Yes I think simulations are helpful visual tools for when you are explaining something and cannot leave the school; Yes when appropriate; Yes, but only if I could change the things listed above (1. Make the size of agents changeable after they're created 2. Simplify the behaviors.); Yes, especially for upper grades; Yes, I think allowing students to visually see a process increases their understanding (ex. seeing a volcano erupt helps them to understand the process involved); Yes, I would use it to introduce a lesson for the students or have the students use it to create a final project; Sure. Some simple simulations this works well; Yes, I wouldn't have to create the simulations. Maybe students could also use this program to explain events in books, grammar usages, and or to tell stories.

However, some of the responses were a little less positive about using this type of tool in the classroom: I would be willing if I had extensive training of how to create things that are eye catchy and interest gaining; If I could not get the proper training I do not think that I would use this software personally; I do not feel very comfortable at this time in using it. Maybe after I worked with it more then I would use it. I could explain something that had already been created though; If I had more time to study the program and felt more confident about the capabilities.

In general, the teachers in the SimBuilder group shared positive opinions about future use: Yes, I would use this tool however; I would suggest having training with an experienced user; Yes, if I was teaching an earth science class. This would prove extremely useful for students who have poor rote memories but have excellent visual/spatial-induced memories; I would be willing if I had the proper support technician or person that I could call to ask questions. With this software you really have to know it well and be about to teach others not to be frustrated with the activity if they do not get it on the first tries; Yes; Yes, in science or history; Sure, this would be good for young

students, since many of them are computer- and video-game-savvy. This would be a good tool for them to use to understand basic science principles.

13. What ideas do you have for simulations that would be useful for you or someone in your discipline? The rationale for this question was to get simulations ideas from teachers during this post-questionnaire brainstorming session. Most of the teachers gave several ideas for interesting simulations as summarized as follows:

Simulation Ideas from AgentSheets Group

- Migration patterns, government evolution and development, timelines for various subjects: etc.
- Use a simulation on a PowerPoint presentation or have children design simulations of what they think will happen
- None at present (not yet teaching)
- Nitrogen cycles, water cycles, metabolism, growth, respiration, photosynthesis.
- Science experiments would work well,
- None at this time.
- Photosynthesis process for plants, volcano erupting, earthquake destruction, erosion, oceanography,
- Chemistry (atoms, elements etc)
- Growth of cities, changes in voting patterns, other social studies related activities.
- Butterfly migrations
- Recreating scenes in plays and novels, creating grammar lessons, giving opportunity for students to create stories.
- Having some type of simulations that would work besides the sciences.
- Seasonal changes weather changes precipitation

Simulation Ideas from SimBuilder Group

- I think this is a great way to simulate science material. The program is perfect to illustrate reactions of the interaction between objects.
- This simulation program would serve useful in explaining military history in social studies classes--especially military events that have been influenced by the weather.
- N/A
- Real life situations
- War simulations for world history... geography simulations
- Science for grade schoolers, maybe even simulations for older students involving more advanced science (physics, chemistry).

14. *Can you think of any changes or enhancements to this system, especially ones that would make it more useful in creating simulations for novices? Please briefly describe the features that you think are needed in building simulation software.* Most of the teachers working with AgentSheets offered ideas for changing the system: Maybe instead of using the word "agent" for the folders, using "New Object" or something, agent is confusing to normal people; Provide easy-to-understand, step-by-step instructions. Include visual aids & graphics; It needs to be easy for the user. Have lots of pre-made simulations to borrow behavior from; Have clipart types drawing available so people would not feel they have to draw; Multi-step process that involved many menus may be difficult for novices perhaps, everything could be incorporated in one or two menus or even just one large worksheet; Enhance the tools and have more animation tools; More drawing flexibility, bigger agents. Sound (I don't know if it has is). Text in the simulation; Be aware that people learn in different ways. Some can be given directions, read through them, and then produce what is there. On the other hand there are others who prefer a more interactive environment; I would allow more room for adjusting the agents appearance.

Most of the teachers working with SimBuilder also offered ideas for changing the system: I only think that more detailed instruction on how to work each function would facilitate the user; The necessary features for building simulation software would be the ability to create objects (paintbrush) and the accessibility to rule making in order to create the movement; The main features that need improvement is the menu diagram as it proved too complicated and more simplified instructions; Screenshots for what people will see and how they would use each tool for each activity and object would be useful; More system help notes.

15. *Any final comments about your experiment activities or the software?* Many participants took this opportunity to make one or more final comments. These are summarized as follows:

Final Comments from AgentSheets Group

-Overall, it was fun, however, it highlighted how far software governing simulations have to go in order to recreate the visions that lurk in the human mind. Thanks for the experience.

- Overall a great idea - just a little tweaking & it will be great :)
- No
- Automatic saving of each newly modified simulation so that when you reset it, it doesn't mess up.
- It was a fun experience that I would like to use with students in the classroom
- I thought that this was a useful tool that needs improvement. If there can be some kind of internet base of section added to this to incorporate the internet into the use of the software—teachers could create simulations using graphics from the internet.
- Students would enjoy creating simulations, creating simulations also involves using problem solving skills and exposure and usage of the computer
- Good Luck with your project
- It was pretty frustrating getting started but its on the right track
- Thank you! Although I do not have much computer experience, with a little trial and error and practice, I could use this more fluently. It is a nice program, especially for middle school and elementary students. I think high schoolers could use it too. It would be a nice change of pace for the students.
- Thanks for letting me participate, sorry if there were any problems!

Final Comments from SimBuilder Group

- I know that I would not have been able to complete the assignments if I had not had someone guiding me through the steps for the first assignments. After practicing though,
- I was able to create a simulation on my own rather quickly.
- This simulation software has great potential if changes in the menu programming and instructions take place.
- This was a very challenging experiment for me.
- n/a
- It was fun!
- Sometimes using the SW was frustrating and seemed a bit difficult, but w/ a little time,
- I began to understand how to use the system, despite the learning curve, I believe it's a good tool for science education.

We found that users did indeed enjoy creating new objects to express a concept, but had more trouble when it came time to make it more meaningful and display some actions. Users of AgentSheets were generally unsatisfied with their drawing tool and many felt it was too rigid and created blocky looking images. AgentSheets users also had a hard time keeping track of the condition and action menus, which with visual rule syntax problems made their experience less enjoyable than the uses of the SimBuilder environment, but there are still usability enhancements that should be incorporated to improve novice usability.

In studying user insights about the tutorial we discovered many things that would improve the tutorial. With the AgentSheets tutorial we had gone through several iterations and with the SimBuilder tutorial this was the first iteration and fortunately there were few problems with the SimBuilder tutorial. Because of the familiarity of the creators with the environments and intent of the simulations a few key explanations omitted from the tutorial, but were added to our script so that in certain situations we would verbally instruct the user in the intent of the activity and appropriate visual programming techniques in these environments (For example, when the user begins to create the volcano model, we would have to remind each user that an agent or object should be created as an independent entity, if they expected it to interact with other objects).

A few participants were longing for a more systems approach to training, but the majority were happy to learn by example and thought it a helpful technique. Most found drawing fun and seeing their creations come to life as personally rewarding. One major concern was with size and blocky look of AgentSheets graphics, and problems with rule visual syntax and many users had to learn through trial and error the proper syntax. Overall the tutorial was helpful, but with this large amount of tasks should be broken into two shorter sessions to maximize results and not lose user focus. We should enhance the tutorial by adding more general visual programming instruction (e.g. more practice with dragNdrop and visual manipulation of interface), add audio cue to support varied learning styles, make sure directions and graphics are clear.

In working with teachers it was very interesting to see how clearly teachers understand the benefit and necessity of giving their video-game-savvy students visually stimulating material, their willingness to give these environments a try in the classroom and their creativeness in suggesting tons of interesting simulations. Developers of educational media and applications for the classroom should definitely utilize teachers' input to greatly improve the environment's usability by reminding us that copy-paste example based reuse is very helpful, and less complexity is better when supporting novice programmers.

6.3.5.7 User Simulation Quality: Informal Observations

As a final set of comparative measures we evaluated the *quality* of the simulations produced by users of the two different systems. Across the learning and reuse sessions, each participant produced three different simulations: a volcano, an ocean with waves crashing on the beach, and a photosynthesis model. In this section we first review the products created by the participants in an informal fashion; in the next section we present a more systematic review provided by expert evaluators.

Figure 6.14 contains two examples of volcano models that we judged to be good representatives of the simulations created during the learning sessions of AgentSheets users. This judgment is based on the combination of the visual look of the world created as well as the appropriateness and sophistication of its behavior (rules). In the example on the left, a volcano is positioned on top of a lake (in AgentSheets this is a single character in a grid). The volcano produces smoke puffs, and lava. The smoke rises into the sky and the lava flows downward. In the second example, the volcano produces lava, which moves downward. There are also some birds present in the model that are flying. Most of the rules created for the volcano project are relatively simple, those we have earlier characterized as basic (i.e. a rule that just has one function like moving in a certain direction.) There were a few intermediate rules (i.e. rules that contain basic functionality plus perform some agent interaction). In both of these examples the volcano object produces another object, which requires object interaction (an intermediate rule). Most of the simulations contained just one intermediate rule, a few objects with basic rules, and decorative background objects.

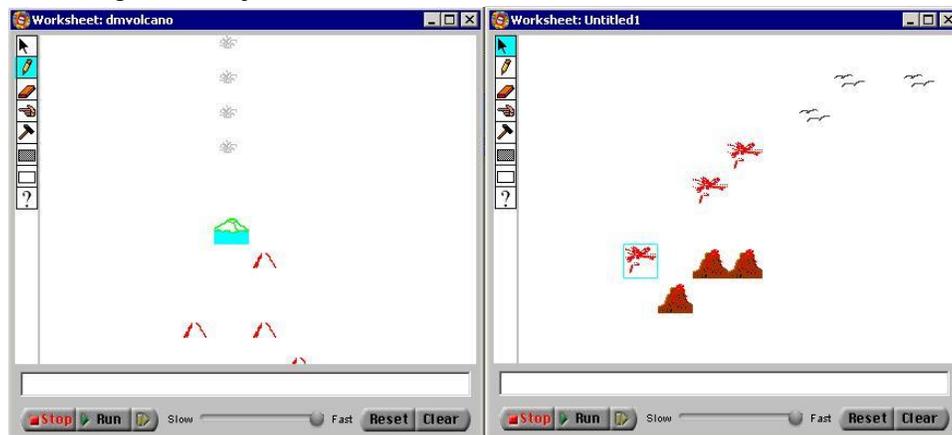


FIGURE 6.14. LEARNING SESSION ARTIFACTS

Figure 6.15 provides contrasting examples of volcano models, in this case weak or poor solutions to the volcano problem. The model on the left also uses visual layers to depict the volcano. The user created the volcano shell, lava to fill its center, a top for the volcano to blow, and lava eruptions. The user saw each of these things happening in the same space, which is a good technique to mimic realism, but the user programmed no interaction among characters that would cause the top of the volcano to blow off—it is caused by the start of the simulation. Lava is produced by the volcano, but the user was not able to integrate this lava-creation rule into the volcano model as a whole, thus the rules do nothing. In the second volcano model, the volcano is just a visual representation or landscape, with no working rules at all. The user was not able to visualize or associate actions with the corresponding rules.



FIGURE 6.15. LEARNING SESSION VOLCANO ARTIFACTS

Examples of good models created by SimBuilder participants appear in Figure 6.16. In the first example, the volcano is positioned on top of an ocean and earth. Pressure builds from the ocean and when it reaches the volcano top, the top blows off and lava, boulders, sparks, and soot are created from the volcano. There are also other objects, for example, birds that fly and clouds that float. In the second example, the volcano produces lava, which moves downward; when it comes into contact with the earth it becomes scorched earth, and also creates steam, sparks and lava from the volcano. The landscape includes birds that fly and clouds that float. Like the simulations built by AgentSheets users, many of the rules created by SimBuilder users exhibited this rather simple behavior that we have characterized as basic (e.g. a bird moves forward by 5). There were a few intermediate rules (e.g. if lava sees earth, then the earth becomes scorched earth). In both examples in Figure 6.16 the volcano object produces another object, which requires object interaction (an intermediate rule). Most of the simulations contained 1-2 intermediate rules, 1-2 objects with basic rules, and decorative background objects.



FIGURE 6.16. SIMBUILDER LEARNING SESSION VOLCANO ARTIFACTS

Figure 6.17 is a contrasting example of SimBuilder models that are relatively weak solutions to the volcano problem. In the first example, a volcano creates lava. The lava object flies upward. In the second volcano model, the volcano produces lava, which moves downward. Both include steam, sparks, lava, and a blue cloud as part of the landscape. However, all of the rules created were basic rules (e.g. lava move forward by 5). These rather simple simulations contained 2-4 objects, with basic rules only, along with a few decorative background objects.



FIGURE 6.17. SIMBUILDER LEARNING SESSION VOLCANO ARTIFACTS

The simulations in Figure 6.18 illustrate high quality solutions to the ocean world problem produced by AgentSheets users including at least one intermediate rule and a recognizable representation. The model on the left presents a scene in which the ocean creates waves (intermediate), waves move toward the beach (basic) and when they reach the beach it causes an erosion of sand (intermediate). The model on the right the ocean also creates waves (intermediate), waves move toward the beach (basic) and the sand disappears when contacted by waves (basic).

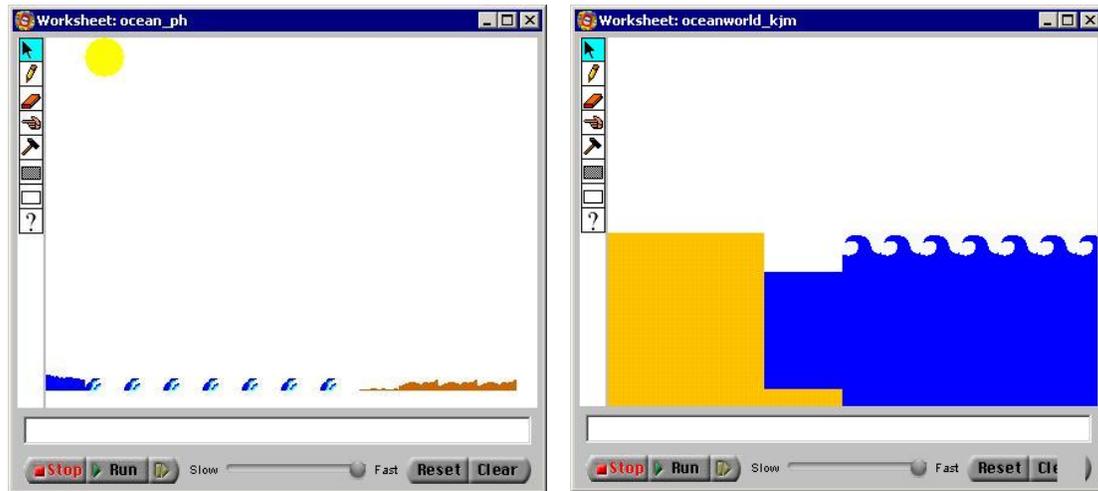


FIGURE 6.18. BEST EXAMPLES OF AGENTSHEETS OCEAN REUSE MODELS

In contrast, the two examples in Figure 6.19 illustrate relatively weak examples of the ocean model. The model on the left has three objects (i.e. ocean, sand, and wave), and the wave object moves (basic). The depiction on the right also includes 3 objects ocean, clouds and sun and has no behavior.

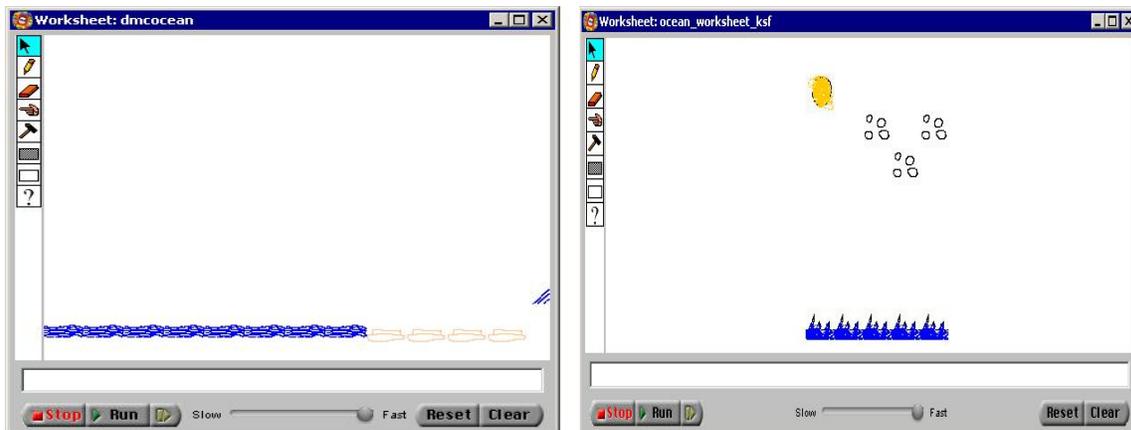


FIGURE 6.19. WEAK EXAMPLES OF AGENTSHEETS OCEAN: REUSE MODELS

The artifact in Figure 6.20 illustrates a high quality example of the photosynthesis model created in AgentSheets. In this model the user created 5 objects. The sun with produces sunrays (intermediate), the sunrays, which fall to the ground (basic), small trees that when contacted by sunrays change to large trees (intermediate) and the ground as part of the background scene.

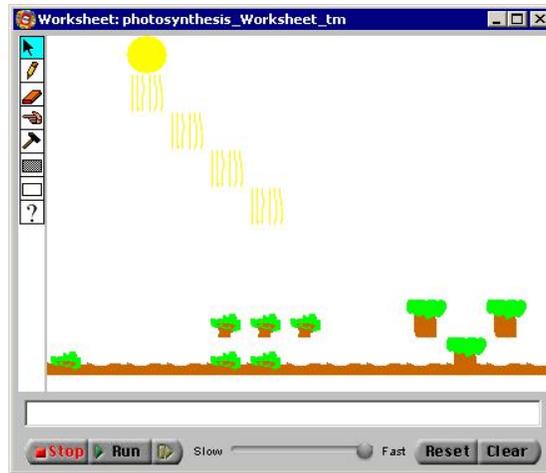


FIGURE 6.20. BEST EXAMPLE OF AGENTSHEETS PHOTOSYNTHESIS MODEL

Figure 6.21 illustrates a weak example of the ocean model created in AgentSheets. This model contains four objects that include sun, sky, clouds and trees, but these objects have no functionality.



FIGURE 6.21. WEAK EXAMPLE OF AGENTSHEETS PHOTOSYNTHESIS MODEL



FIGURE 6.22. BEST EXAMPLES OF SIMBUILDER OCEAN MODELS

The artifacts in Figure 6.22 illustrate good examples of ocean models created in SimBuilder. The first model contains 4 objects: ocean (basic), beach (intermediate), big cloud (basic) and little cloud (basic). The basic objects just move and the beach moves when contacted by the ocean. The second model contains 5 objects; ocean (intermediate), wave (intermediate), mist (basic), beach and sun as scenery. The basic objects again just move and the ocean creates waves and the wave creates mist and disappears when it hits the beach.

In contrast, the two examples in Figure 6.23 illustrate weak examples of the ocean model created in SimBuilder. The first model contains three objects, and only the wave moves. The second model has three objects; the ocean emits waves (intermediate) and the waves move (basic), but the visual representation is poor.



FIGURE 6.23. WEAK EXAMPLES OF SIMBUILDER OCEAN MODELS



FIGURE 6.24. BEST EXAMPLES OF SIMBUILDER PHOTOSYNTHESIS MODELS

The artifacts in the Figure 6.24 illustrate high quality examples of photosynthesis models created in SimBuilder. Both models present a good visually representation of the effect they intend to model and have good use of intermediate rules. The first model contains a sun creating energy waves (intermediate), energy waves that move (basic) and a small tree that when contacted by energy changes into a larger tree. The model on the right also contains a sun, which creates sunray (intermediate), sunrays, which move (basic) and a flower that grows into a larger flower when contacted by sunrays (intermediate).

In contrast, the example model in Figure 6.25 is a weak example of the photosynthesis models created in SimBuilder. In this model the user created 4 objects a sun, sky and grass as scenery with a sunray that moves (basic). The concept the user is trying to depict is not very easy to recognize.

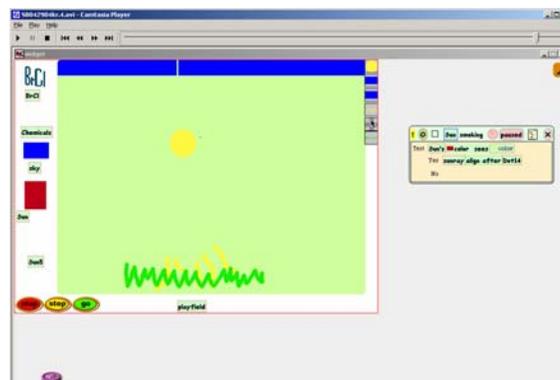


FIGURE 6.25. WEAK EXAMPLE OF SIMBUILDER PHOTOSYNTHESIS MODEL

6.3.5.8 User Simulation Quality: Expert Ratings

As a final evaluation metric, we recruited five user interface experts to judge the quality of the simulations created by participants in their learning and reuse sessions. The expertise of these judges was based on their highest degree in computer science, experience in teaching and learning or related fields, experience in software engineering or design, and experience in usability engineering, user interface design, interface evaluation or graphic design.

In order to ascertain whether interface evaluators would find the SimBuilder creations of higher quality than the simulations created in AgentSheets we devised a User Interface Evaluation questionnaire that gathered experts' background along with their ratings of the action and visual representations of sample simulations. Candidate experts were identified through consultation with committee members and other colleagues and contacted by email with a brief description of the task requested. After agreeing to participate, each evaluator was emailed the evaluation questionnaire (Appendix O). The first section of the questionnaire gathered background information to validate the participant's predicted expertise (e.g., questions probing experience software engineering and design, usability engineering, design, and user interface evaluation). The second section of the evaluation introduced the evaluation task and informed the experts that the evaluation was to ascertain the aesthetic quality of these artifacts and was to be graded with two individual ratings, one of visual representation and an action rating. The third section consisted of a rating guideline for consistency, which gave the evaluator concrete examples of anticipated ratings (Table 6.29).

Table 6.29.

Evaluation Rating Guidelines**Center for HCI: User Interface Evaluation Rating Guide**

Ratings	Action Representation	
5	Excellent	A rating of Excellent indicates that the user models the phenomena with a variety of relevant working objects (5 or more).
4	Above Average	A rating of Above Average indicates the user has fairly realistic depiction with at least 4 working relevant objects.
3	Average	A rating of Average indicates the user has a realistic depiction with at least 2-3 objects working.
2	Below Average	A rating of Below Average indicates the user has a at least one object working.
1	Poor	A rating of Poor indicates that the user model has no working objects.

Ratings	Visual Representation	
5	Excellent	A rating of Excellent indicates that the user had good realistic drawing that models the phenomena with a good use of color and 5 or more objects.
4	Above Average	A rating of Above Average indicates the user has fair use of color and realistic depiction with at least 4 relevant objects.
3	Average	A rating of Average indicates the user has fair use of color and realistic depiction with at least 3 relevant objects.
2	Below Average	A rating of Below Average indicates the user has a fair use of color and realism with at least 2-3 objects.
1	Poor	A rating of Poor indicates that the user vaguely conveyed the model with 2-4 objects.

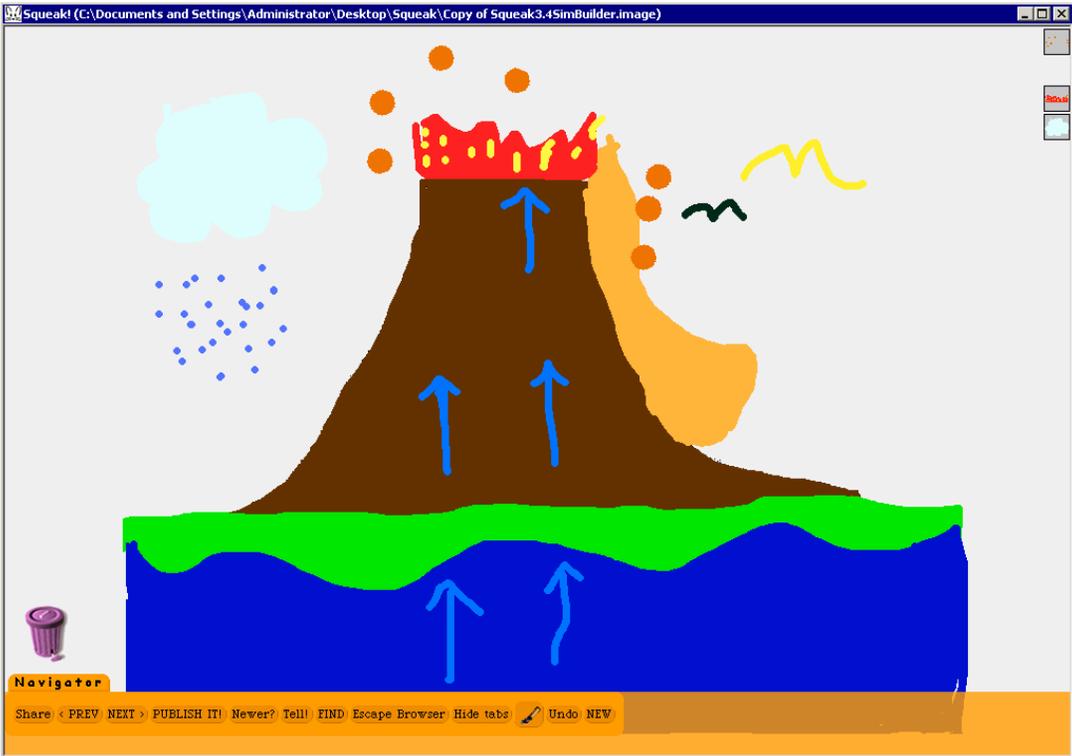
The rating guideline provided a well-defined criterion for assessing whether the artifact expresses the problem model (i.e., a volcano, photosynthesis, an ocean). The balance of the evaluation consisted of 40 images of user-created simulation screenshots. These examples were selected used the Purposeful random sampling method instead of presenting all of the simulations (i.e. 57 user created simulations, which would have been expanded to 96 simulations with the initial experimental design) that we collected in an effort for brevity and to be less time consuming for expert evaluators. “For many audiences random sampling, even of small samples, will substantially increase the credibility of the results” (Patton p.178). The experimenter sets up a random procedure for selecting simulation results. We would present the simulations in random order not to bias the user with an alternating selection process where the user might be able to

determine that every other representation was related by system. The evaluation would randomly select a simulation based on type of simulation volcano, photosynthesis and ocean world and continued this selection until we had selected 40 images

The 40 images were presented in a random order to minimize effects of presentation order on judgments of AgentSheets versus SimBuilder models and good versus bad examples. We asked the experts to print the entire document or the rating guidelines to refer to during their ratings. They were asked to provide a rating either on hard-copy by circling their selection within the document, or by typing their responses into the digital form of the questionnaire. More detail can be found in Appendix O.

Our experts were given information about the intent and rationale for the evaluation, namely to ascertain the perceived effectiveness of the teachers' simulations: "We are not trying to find out the quality of the artists, but the perceived quality of the artifact (i.e. being a good representation of the desired model)". We also informed them that the simulation models were addressing topics relevant to middle school physical and earth science. The models to be evaluated were summarized as follows: volcano model, photosynthesis model, and ocean model.

The following is an example of an artifact to evaluate: End-user created.



Volcano
Volcano with lava, explosions of boulders, ashes, and pressure that rises to cause eruption. Water and earth surround the Volcano and a bird flies to escape the volcano

Visual Representation Rating
Select the response that best indicates your rating.

- 5 Excellent**
- 4 Above Average**
- 3 Average**
- 2 Below Average**
- 1 Poor**

Action Rating
Select the response that best indicates your rating.

- 5 Excellent**
- 4 Above Average**
- 3 Average**
- 2 Below Average**
- 1 Poor**

FIGURE 6.26. USER INTERFACE EVALUATION VOLCANO EXAMPLE

Our informal judgments had suggested that the simulations created using SimBuilder were more sophisticated, with a more aesthetically pleasing and natural look than the grid-based representations created with AgentSheets. The objects in the SimBuilder models had varied sizes and more naturalistic visual features. As the mean values for the rating scales suggest (Table 6.30), the expert panel seems to confirm these informal observations. In this table, the ratings for all problems (volcano, etc.) have been averaged for all five judges but broken down by SimBuilder versus AgentSheets tool. All ratings were on a 5-point scale from 1 = Poor to 5 = Excellent.

Table 6.30.

User Simulation Quality Ratings by Expert Judges

Model Rating	SimBuilder Mean (SD) N=19	AgentSheets Mean (SD) N=19
Visual quality*	2.42 (0.42)	2.18 (0.53)
Action quality*	2.50 (0.29)	1.93 (0.54)

5. Excellent 4. Above Average 3. Average 2. Below requirements 1. Poor

* indicates $p \leq .10$ for the mean difference contrast

Although neither SimBuilder nor AgentSheets were judged to be of high quality (not surprisingly for these novice productions, the highest average is still below the “Average” rating in terms of meeting requirements), the judges did seem to have a general preference for SimBuilder solutions. Each evaluator was given 7-8 examples of the volcano, ocean and photosynthesis models for a total of 22 models to evaluate for visual quality and 22 models to evaluate for action quality. Our statistical testing reveal that both of the test mean differences were statistically significant: the visual quality of all models [$F(1,18)=319.59$, $p < .01$]; and the action quality of all models [$F(1,18)=244.63$ $p < .01$]. We attribute these findings to the greater variety of rules available for users to choose from in the SimBuilder rule creation toolkit, the convenience of rule behavior from one simulation to the next, and the richer tools for drawing the objects making up the simulations. The user could easily copy a rule, which is placed in the paste buffer to be reused in a subsequent simulation or change the name and look of an object and reusing its predefined behaviors. Post-hoc comparisons

indicated that these differences were consistent across all three model problems (volcano, photosynthesis, and ocean).

6.3.6. Summary of Comparative Evaluation

The purpose of this research was to create a framework to provide improved support for novice programmer teachers in their creation of educational software simulations, and to validate this work we compared the SimBuilder environment with a state-of-the-art commercial system, AgentSheets. In this comparison we were interested in four general hypotheses regarding learning, reuse, programming style, and expressivity. In this section we summarize the findings of the comparative studies (both analytic and empirical), both by re-visiting the four sets of hypotheses, and by considering the findings a more general level.

6.3.6.1 Learning

The first research question was “*What are key learning challenges for teachers working with visual simulation environments?*” Relevant results include the analytic studies of AgentSheets and SimBuilder, as well as the empirical results related to initial learning and satisfaction.

From the usability inspection we found that it was easy to create objects in both environments, but in the AgentSheets environment we had a problem with overloading of the drawing tool operation. In order to add an object to the workspace, users were required to reselect the drawing tool and object to add it to the workspace; this is a nuisance when the analogous task is handled in SimBuilder an automatic consequences of drawing an object. Also the single bit editor provided in AgentSheets did not provide the robust drawing functionality users were accustomed to (e.g., in Microsoft Office), where the SimBuilder drawing tools were very satisfactory. With rule creation we found that we were able to use the rule creation toolkit to add behaviors, but the visual syntax of AgentSheets would cause slight problems that had to be debugged.

During the learning session, we observed that users were able to understand how to manipulate objects in world, make changes to existing rules, create basic objects and

add behaviors to their creations. The major usability problems of AgentSheets users stemmed from the dual functionality of the drawing tools, which confused users causing them to repeat the same tasks many times; with use of fill tools in many instances users had to recreate their objects when bad fills destroyed their creations. SimBuilder users were in contrast quite satisfied with the drawing tools it offered. In using AgentSheets participants were also forced to negotiate many windows and from time-to-time were confused during their interaction with the user interface and specifically the rule creation toolkit. The SimBuilder users only had to deal with one window for rule creation, which simplified their rule creation tasks and avoided that confusion.

With respect to specific hypotheses related to learning, one prediction was that *there would be a significant improvement in teachers' overall satisfaction with SimBuilder vs. AgentSheets*. Quantitative evidence related to this hypothesis can be found in teachers' responses to the general user satisfaction questions in the post-test questionnaire. Although the results were not strong, we did find a pattern of reactions suggesting a more favorable reaction to SimBuilder than to AgentSheets. Specifically all but one of the items showed mean responses in favor of SimBuilder, but only one difference was close to significant (Rigid-Flexible).

A second prediction was that *there would be a significant improvement in teacher's satisfaction with the ease of use, motivation to use, and fun in using SimBuilder vs. AgentSheets*. This prediction of course is very similar to the previous one but points to more specific consequences of the new tool.

Quantitative data related to this hypothesis is found in responses to the post-test rating scales that were designed to assess these specific aspects of users' experience (see Tables 6.24 – 6.28). As for the general satisfaction measures, the means tended to favor SimBuilder but generally the differences were not statistically significant. One exception to this was in the area of motivation, where three of the four items showed a reliable advantage for SimBuilder and the fourth approached significance. Thus it seems that the major consequence of the new tool with respect to users' satisfaction was in the area of

motivation, fun, enthusiasm, and so on. We find this to be a very promising result, as having the motivation to design and create educational simulations is absolutely essential among overworked professionals like K-12 teachers.

These quantitative findings are complemented by the analytic evaluations of the two environments. For example, the comparative claims analysis (Table 6.6 and 6.8) highlighted the support provided by SimBuilder for an enriched toolkit for drawing objects, creating rules, and more specifically the ability to create rules that enact random behavior. This increased flexibility was claimed to add to the creativity and resultant “fun” of using SimBuilder to create simulations. In fact, user comments in the post-test questionnaire are consistent with these expectations, with AgentSheets users often complaining about the drawing tools provided.

A third specific prediction related to learning challenges was that *there would be a significant decrease in teachers’ learning time for SimBuilder vs. AgentSheets*. The quantitative evidence for this was provided in the comparative analysis of session times for both the initial training phase (water cycle) and the performance phase (volcano). The statistical analysis revealed a modest advantage for SimBuilder users, although a follow-up analysis of covariance indicated that at least some of this advantage was due to the greater computer experience of users in the SimBuilder group. Furthermore, as users became more familiar with the environments (e.g., moving on to their reuse projects), the time advantage for SimBuilder disappeared. Because the individual variability also increased as learners became more self-directed, it is difficult to know whether the advantage truly disappeared or simply became masked by the larger effects of individual variability.

A fourth prediction regarding learning was that *there would be a significant increase in the number of objects and rules created by teachers using SimBuilder vs. AgentSheets during the learning phase of the comparative study*. The quantitative data related to this hypothesis was gathered by counting the objects and rules created during the learners’ simulation project (the volcano). Analysis of these data revealed modest

evidence of an advantage for SimBuilder, namely an increase in number of basic rules that approached significance. In this case, the difference was unrelated to users' computer experience. Although this individual contrast was not a strong effect, again the general pattern suggested an advantage for the SimBuilder user group.

6.3.6.2 Programming Style

The second general research question to investigate is "*How can we assist teachers in learning and doing visual simulation programming?*" The approach was to contrast the programming style supported by AgentSheets versus SimBuilder. For example, in Chapter 3 we reported that prior work on object-oriented programming has claimed that objects are a "natural" way to decompose real world problems into computational solutions (Rosson & Alpert, 1990). Thus we proposed an object-oriented simulation framework to address some of the problems with current rule-based visual simulation environments because it enables users to adopt a problem-oriented conceptual focus instead of a visual animation focus. Thus our general prediction was that the object-oriented message-passing programming style incorporated in the SimBuilder environment will facilitate simulation construction relative to the animation-oriented graphical rewrite rule programming style of AgentSheets.

A specific prediction associated with this question was that *there would be a significant difference in teachers' satisfaction indicating good support for novice programming (i.e. easy to remember tool location, easy object and rule creation, and increased user confidence that they can create a working simulation)*. The quantitative data related to this prediction can be found in the subjective ratings responses related to programming style (Table 6.27). The data for these items suggested mixed reactions, but none of the differences between SimBuilder and AgentSheets were significant. This underscores the importance of the motivation effects reported earlier; it appears that at least for these users, how they feel about doing the programming may be more salient than how complex or comprehensible the programming is.

6.3.6.3 Reuse

The third research question to investigate is “*What forms of reusable material are suitable for teachers building simple science simulations?*” The approach was to investigate generic components and concrete examples as reusable material during simulation construction sessions. In Chapter 3 we reported that years of research have shown that humans learn well from examples. However, it is not clear at what level of granularity such examples are most useful. The preliminary studies forming our requirements analysis suggested that teachers might be better able to reuse generic components (e.g. a “mover”) than a piece of a concrete world (e.g. a “waste chemical molecule”). For the generic components, they may avoid the problem of investigating the behavior it encapsulates, reducing the cognitive overhead in choosing a reusable object.

A specific hypothesis associated with reuse concerns was that *there would be a significant improvement in user-reported understanding of generic components vs. example components when used to build new simulations*. The quantitative data related to this prediction can be found in the subjective rating scales related to reuse, namely those that compared users’ understanding of the ozone world versus the starter world (Table 6.21). We reported that users felt more confident in their understanding of how to reuse the generic components versus the concrete examples, but this confidence did not seem to extend to their understanding of the detailed behavior of these reusable worlds.

A performance-oriented prediction related to the reuse question was that *there will be a significant decrease in teachers’ reuse time for using components versus examples as reusable material*. The quantitative evidence related to this hypothesis can be found in the session times for the two reuse projects organized by the type of reuse material provided (Table 6.20). Unfortunately despite the fact that the mean times for component-based reuse were “lower”, the statistical analysis revealed that the difference was not reliable. Thus we have no quantitative results in support of this prediction.

Because we had noticed that differences in session times became very difficult to detect once users were working on their self-directed projects (i.e., due to immense individual variability), we also analyzed the numbers of rules and objects created in the two different types of reuse projects. The quantitative data for this analysis can be seen in Table 6.23. The statistical analysis of these data indicated that none of the general differences were significant but there was a somewhat curious interaction between reuse type and system. We observed that AgentSheets users seemed to benefit more from the generic components (in this case creating simulations with more rules), but that SimBuilder users did better with the concrete example world.

From a qualitative perspective, there is a pattern of user comments that suggest an advantage of the generic components, although it is difficult to draw strong conclusions from this. For example, in post-test questions asking specifically about the reusable materials, users in both groups commented on the usefulness of the starter world object (e.g., the emitter object was seen as particularly useful), but no users commented on aspects of the ozone example world. Of course it is possible that the generic components are simply easier to comprehend and remember in such commentary, but even this as an explanation is promising with respect to our initial hypotheses.

6.3.6.4 Expressivity

In evaluating a work that is very subjective like design and creation we need assess the expressivity of artifacts crafted. In Chapter 3 we noted that expressivity is a highly qualitative measure and as defined by Merriam Webster (www.m-w.com) is the quality of being expressive, conveying a meaning or feeling. In simulation design, when we speak of expressivity we are alluding to whether the designer's creation realistically expresses a real world model: we want to judge the expressive quality of the artifacts and the form imposed by their environments. We organized a subjective assessment by user interface experts to consider the expressivity of simulations produced by users of the two programming environments.

A specific prediction associated with this concern was that *objects and simulation worlds created in SimBuilder would be more sophisticated and have a more aesthetically*

pleasing and natural look than those created with AgentSheets. One piece of quantitative evidence in support of this prediction concerns the measures of artifact complexity summarized earlier (objects and rules used to build the various projects). We found little evidence of an advantage for SimBuilder simulations, although there was a tendency for the SimBuilder volcano projects to contain more basic rules than the AgentSheets volcano projects. It seems likely that these users were still quite novice, so hoping to find differences in complexity for users at this level may be premature.

Another source of evidence concerns the ratings provided by the panel of five experts. These ratings assessed both the visual design and the behavior of the projects created by users in both groups (Table 6.26). As the statistical analysis of these ratings indicated, the judged quality of both the visuals and the actions of SimBuilder simulations was greater than those created in AgentSheets. These findings suggested that the SimBuilder drawing tools (the variable size of the objects, the more natural fill tool, convenient short-cuts for managing color, etc.) encouraged users to be more expressive in their visual design than those use the more limiting tools of AgentSheets. The higher quality action ratings may be due to users' improved ability to focus on the behavior they were trying to achieve and not have to worry about managing multiple windows, palettes, cursors, and so on. The fact that the experts found the SimBuilder products more pleasing is also consistent and reinforces the related finding that the SimBuilder users were more enthusiastic about their programming experience.

6.3.6.5 Summary

The analytic and empirical work reported in the chapter is rich and complex. The empirical results are not strong, but a number of converging analyses with modest findings point to a general advantage of SimBuilder over AgentSheets, at least with respect to the initial learning about the environment, and with respect to the complexity and aesthetic quality of the simulations produced. These findings are consistent with the analytic work that guided the design of SimBuilder and that subsequently compared it to AgentSheets. Clearly there were large individual differences among users, and the size of the samples used in the empirical study made it difficult to obtain powerful tests of the

results. Looking across all of the findings however, we are confident that the design of SimBuilder has been a useful step in making the learning and use of simulation tools more accessible and engaging to teachers.

CHAPTER 7. DISCUSSION AND CONCLUSIONS

This research project was initiated with informal usability inspections of more than two dozen environments that utilize visual end-user programming techniques for simulation software, which resulted in a taxonomy to illustrate their features (see Appendix A). We followed this initial survey with a more detailed analytic evaluation of specific issues related to the general user interface, the drawing tools and the rule creation mechanisms in five environments (see Appendix B). We also carried out detailed empirical evaluations of two different visual simulation environments. The first of these two studies was with teachers using AgentSheets for learning, creating and reusing science simulations; the second evaluation was with middle school students training them to program in a direct manipulation environment and testing the knowledge they retained from the experience. Both of these studies provided a basis to ascertain which features and implementation style would be more useful for our novice programmer participants.

The general aim of the preliminary research was to develop a better understanding of the requirements for visual simulation construction by studying existing systems. These requirements were used to motivate and guide iterative development of a simulation tool (SimBuilder) that might better support novice users in simulation creation. The usefulness and usability of this new environment was contrasted to that of a very successful commercial tool (AgentSheets) in a series of analytic and empirical evaluations, and while the findings involve a mixture of qualitative and quantitative results and thus are difficult to summarize in a simple fashion, there were many sources of evidence suggesting that the new environment presents a number of advantages for teachers who have no programming experience (usability inspection, user comments and ratings, expert user ratings, etc.). Beyond the scope of the dissertation, the future work suggested by this research includes questions of teacher adoption of tools like SimBuilder in their classroom settings. In this chapter I review the findings of the research, give its scientific and practical contributions, discuss future research, and conclude with a summary of the significance of the studies reported here.

7.1 Results Summary

The research addressed questions related to teachers as novice programmers in several different areas. First we wanted to understand the obstacles for teachers learning to use visual simulation environments, and subsequently whether and how a simpler graphical user interface—but one that still relied on direct manipulation—could address these obstacles. Second, we were concerned about the usefulness of the graphical rewrite paradigm when attempting to construct realistic simulations (e.g., with differently sized and behaving characters); we wanted to see what the impacts of an object-oriented framework would have on behavior programming. Third, we were very interested in helping teachers to develop simulations more quickly by providing reusable materials, but wanted to know at what level of granularity (full examples or generic components) these materials would be best understood and adapted to specific programming tasks. Finally, we knew that the quality of the resulting simulations would be important to teachers' motivation and subsequent use of them in their classes, so we wanted to explore the impacts of different programming environments on the quality of the products.

With respect to teachers' learning obstacles we found that the complexity of a highly interactive environment caused problems for our users. The AgentSheets environment caused a number of problems for novices trying to create visual simulations. Some of these were basic usability problems, for example, learning to use the drag and drop style of rule construction, while others were more tied to the visual character of the programming language, for example, the problem in layering that occurs when two characters occupy the same visual grid position.

The SimBuilder environment seemed to address some of these problems; for example, the environment provides fewer windows to manage, so the drag and drop style of rule construction is easier to manage. Because it does not use a grid-based layout, users of the new system did not experience problems related to space “sharing” on the screen. Analysis of performance with SimBuilder versus AgentSheets indicated a modest advantage for SimBuilder in learning time for the first phase of users' work, and a modest

advantage in simulation complexity for the second phase of their work. Although general subjective reactions to the two systems did not reveal an advantage for SimBuilder, there was strong evidence suggesting that the teachers who used SimBuilder were more enthusiastic and motivated than those using the AgentSheets system.

With respect to the problems associated with behavior programming, the major obstacle for novices is that of brittleness of visual rewrite rules: rules in graphical rewrite systems are based on exact visual syntax. These rules cannot be reused when the semantics are slightly different. This causes an explosion of rules caused by visual syntax. Users can create basic rules quickly and easily, but would have to create a new rule set for each situation that is slightly different. In our object-oriented solution we reduced this computational complexity with a message passing approach.

In general we discovered other obstacles to learning and building simulations, such as understanding how to design content when beginning with a problem; designing the simulation and building simulation components; the time required to build things from scratch (teachers' time is already heavily taxed so their time for this type of task is minimal); design and analysis is hard; lack of motivation to build or customize materials when textbooks have broad coverage of material and exercises. Many of these obstacles are inherent in the process of building simulations, but SimBuilder seemed to be more motivating for users. This general advantage may help to address inherent problems such as the time required to do the programming work.

For reuse concerns our expectation was that generic simulation components should facilitate object level reuse more than the re-purposing of example simulations. When reusing components, the user has less information to decompose and decipher, creating less cognitive baggage and making it easier to understand the functionality of the component. In contrast, when building a simulation through example reuse, the user first must understand what the components of the simulation are and then what functionality they contribute to the example. Because generic components are designed to provide just one function, the initial comprehension should be eased. The performance data did not

support an advantage of components over examples. However, there was some indication in users' subjective reactions that our expectation was correct: Users felt that they better understood the reuse of the generic component world than the concrete example world.

With respect to questions of the aesthetic quality or expressivity of the simulations produced by SimBuilder and AgentSheets users, respectively, we examined the ratings made by a panel of experts, asking about the quality of both action and visual look of sample simulations. We attribute the more positive ratings of SimBuilder projects to greater ease of use of the tools (teachers didn't have to add first object instantiation to workspace), greater variety of drawing tools (with multiple pixel size tools for drawing and erasing), and the ability to create objects of any size that they wished (they were able to visualize and create objects in a more realistic fashion vs. limit to desktop icon size).

The action ratings were significantly in favor of SimBuilder. We attribute this to the greater variety of rules available for users to choose from in the rule creation toolkit, we also attribute this to greater reuse supported by the environment (objects and rules can be easily copied from one simulation to the next). The user could easily just copy a rule, which is placed in the paste buffer to be reused in a subsequent simulation or to just change the name and look of an object and reusing its predefined behaviors.

7.2 Lessons Learned

In the process of my research there were many lessons; in this section I will highlight a few with commentary and/or example episodes. In an effort to create a software product that was usable by a novice community, we utilized the target population during the every juncture of the experience. Our software engineering lifecycle was not a traditional model like the waterfall model of software engineering, but for rapid prototyping was an expanded version of the task artifact cycle (Carroll & Rosson 1992), which was very useful and flexible. Using an expanded task artifact cycle allowed us to iteratively refine our requirements throughout the entire process, based on our interaction with our end-users.

We included teachers in pilot sessions to identify their level of experience and general computing ability to help us refine our requirements and utilized teachers working with our prototype to improve user interaction for this group. We found some problems that were caused by very simple aspects of our interface, and just a change in interaction technique will be required to reduce and/or prevent user error. For example, to change direction of an object the user must activate a handle that controls direction, this technique needs to be simplified or better instruction must be provided in our minimalist instructional manual

In the area of programming an interactive environment, the biggest problem encountered was with handling multiple processes and threads of execution. The examples that we had seen of the underlying Morphic environment and Etoy examples generally consisted of one object performing. To present a more realistic depiction for a science models we needed multiple interacting objects, the problem ensued that each new object had its own thread and each new thread slowed system processing, and with our emitters creating new objects our system was littered with new threads. The erasers in the environment that we created to assist the novice user was deleting the depiction of objects, but was not deleting the instance it was removing it from our view and storing it in another location, so in effect the thread was still running in background and congesting the system processor, at times crashing the environment, and even bogging down CPU as being at near 100% utilization. Once we identified this problem our simulation ran more smoothly and our CPU was not overly utilized for this one application.

Do you build from scratch or build on top of an already testing infrastructure? With the potential for having a more robust tool, I decided to build upon an established tested environment. In general there are many upsides and downsides joining a large open source community. It is very helpful to work in an open source community and you will find many programmers that are working with the same environment, but unfortunately while doing the majority of this project very few were developing this the Morphic portion of the system therefore I only found one developer that even had an idea of how to address the problem I was encountering, so when I described a problem and

sent it to the developer list no one responded. Only when I happened to meet one of the original developers of the system at SqueakEnd 2002 at Georgia Tech was I able to show him an example problem to express my concerns with Morphic and find an answer to some of my troubles. I found that many of my troubles were related to developing on a different type of machine than the original developers were using. Many of the Squeak developers started at Apple and therefore all used Macs and when I was explaining problems and they were explaining fixes, we had the wrong frame of reference and the fixes weren't working for my environment and they couldn't recreate my error. This has taught me to always find out the environment details when specifying a problem and solution to prevent this headache.

7.3 Contributions

The series of analyses and experiments comprising the dissertation make several contributions to the field of computer science:

1. This dissertation is one of the first formal studies of teachers developing educational software. Teachers were involved at every juncture of this endeavor—giving requirements for simulations that would be appropriate, participating in pilot studies, and learning and evaluating the experimental system.
2. This work studies and critiques many pieces of simulation software, creating a taxonomy of visual end-user programming environments for educational simulations, along with an evaluation matrix contrasting environments.
3. This work established baseline requirements and suggested a framework for visual simulation tools, and demonstrated this framework in an environment that was very attractive for our user group.
4. This work developed and used minimalist materials to train teachers without programming experience—but with content matter expertise—the basic skills visual end-user programming.

5. This work expanded the exploration of component abstraction on reuse usefulness and comprehension, although discovery of the interaction between reuse type and system raises a whole set of new questions.
6. Finally this experiment and dissertation demonstrates the general promise in the study and application of reusable components in simulation construction.

7.4 Future Work

As in our previous work with teachers we had considerable positive outcomes and feedback from teachers (Rosson & Seals, 2001). All of our users were able to successfully create new simulations using both the AgentSheets environment and the SimBuilder environment. They were able to program with the support of direct manipulation techniques, rule palettes, and (in the case of SimBuilder) reusable template objects. The complexity of their creations and their reactions varied widely based upon the users' understanding of the tool and task, but at the end of the experiment, users of SimBuilder displayed significantly higher motivation, enthusiasm, and feelings of fun than those working with AgentSheets. Our expert user evaluators found that the six highest rated user projects were created with SimBuilder.

During the users' training experience, we tried to exploit some reuse strategies of expert programmers in order to train our novice programmer teachers in software reuse techniques. We introduced them to the usage context and encouraged them to assess the similarity of simulation objects and rule behaviors to their assigned task. Then, after studying these bits of context, they would reuse the objects and their behaviors by analogy, by remembering and reproducing the content, or with copy-paste techniques.

Each participant created two simulations, one from each reuse model. We were pleased to see that users were able to reuse both types of material; however we believe that the quality of simulations will improve as our library of reusable simulations increase. We will have to study the possibility of a user finding a simulation that is of very near context to their desired artifact. For example, we observed a tendency for the

emitter object in the starter world to map well to an ocean creating waves, but perhaps not so well to a sun creating energy or rays. This may also be related to the relative concreteness of an ocean and waves versus the more abstract scientific model of sun and energy. This raises questions to address in future research: Will the context-mapping between a reusable model and target project affect the usefulness of the reuse components; will components be attractive for reuse even when the mapping is not close? Even if it is difficult to provide generic components that map to all possible contexts, we are still confident that these abstract components may be effective in training teachers about the common objects and functions useful for earth and physical science models.

The group of components that we presented to our teacher developers was very abstract (mover, replacer, changer, emitter, and eraser). The objects were created in an effort to reduce cognitive baggage. They were also created to support visual programming in a more naturalistic manner by attempting to scaffold the simulation developer in drawing parallels between the object oriented programming and real world object behavior. For future studies, we can refine this set of abstract components and make them less abstract and more tangible to the user (i.e. instead of a mover we would create a wheel to indicate motion, instead of a replacer we would have a magic wand to indicate change, etc.). Once we have a more concrete set of reusable components, we would perform a study to test whether the more concrete reusable components were more helpful to novice programmer developers than our generic abstract components.

In future studies we need a more detailed analysis of areas that address mode of programming style and whether English-like programming or production-rule style programming with condition and actions is more intuitive for novice programmers, therefore our analyses of these issues were not conclusive. With respect to our analysis of reuse by novice programmers, we pose the following rhetorical question. Which operation is easier? Creating new agents and functionality from scratch, or reusing agents and functionality? For both AgentSheets and SimBuilder, users expressed their support for the concept of reuse. Thus it seems clear that providing reusable material is an effective direction to take, but many questions remain concerning the role of concrete

examples versus generic components. There was some evidence that users of AgentSheets benefited more from the components, while SimBuilder users benefited more from the concrete examples. We have speculated that this might be related to the motivational characteristics of SimBuilder, but this finding is certainly worthy of further study. Why would users of an object-oriented programming environment be better able to work from concrete examples, but users of a production-rule environment better able to work from the abstract components? Rather than answering our original question about how best to support reuse, this research has expanded the exploration to also consider interaction effects with the programming paradigm.

We also identified future research issues related to the direct manipulation techniques and techniques for accessing an objects' encapsulated information as well as opportunities for improved error detection, recovery, and notification. For the SimBuilder tutorial in particular, we recognize that its development is still in early stages. Following the minimalist design model (Carroll, 1990) we predict that future revisions of the tutorial will be able to better anticipate and support errors on the part of learners. For example, accessing the Squeak "halo" the first time was a common problem for SimBuilder users, and this can easily be addressed by a tip or recovery hint in the tutorial. A similar example from AgentSheets concerns the continuing difficulty in using the paint fill function; this specific skill might be best supported through an elaborated interaction dialog the first time it is used in the tutorial.

For future work, it could prove beneficial to revise and re-conduct this study with a population of in-service teachers and as a longer-term study, to increase result reliability. It would also be interesting to combine such a study with an investigation of collaboration in simulation construction (e.g., having students create simulations in collaboration with their teachers). Finally, it would be useful to carry out a longitudinal study where teachers and their classes contribute ideas and simulations to populate a library of example simulations that could support learning and reuse within a community.

Ultimately, our hope is to provide software that will be portable and inexpensive. Instead of the teacher needing lots of technical support and a computer for every, our

future work might investigate the use of alternate devices (e.g. a palm pilot) for running or creating simulations. Another possibility would be to give students homework assignments in which they download and modify interactive projects using their home computers.

With the proliferation of the Internet and the World Wide Web, many specialized online communities have begun to emerge. The professional educators' community is no exception. As the Internet has become accessible to more and more people, a number of communities have developed to provide support for teachers. These Internet communities are gathering places designed to support creativity and encourage participation. Some of the communities investigated were Educational Object Economy (EOE), Educational Software Components of Tomorrow (ESCOT), and TAPPED IN at Stanford Research Institute. Some of the largest educational communities examined are EOE and TAPPED IN. These communities invite teachers and developers to collaborate to learn and share their experiences with technology.

Most professions have organizations that promote professional development, but most teachers do not have access to such resources because they both work in isolation in their classrooms and have few other individuals in their content area within their schools. A local effort, the LiNC (Learning in Networked Communities) project, emphasized a complementary view—the need for local community support for teachers [Carroll et al., 2000; Koenemann et al., 1998]. In most professions, there is peer support for professional development activities with face-to-face meetings where many aspects of work and life are shared, including the sharing of knowledge and resources. In teaching, the opposite is usually true. Many schools may only have one or two teachers in an entire school that teach a given grade and content area.

The new online educational communities provide teachers with many opportunities for discourse and educational exchange. As a result, they have been used as organizers of professional development activities. They provide arenas for sharing ideas and building relationships that would normally take place in meetings conducted in person [Schlager et al., 1998]. For example, ESCOT's goal is to become a premiere site

for educators and developers to store and create diverse educational software components that will work together through a common architecture. Every product created in this collaboration will be integrated into the organization's database in the hope that everyone will use a common interface to make the software more universally compatible.

The current focus of ESCOT is mathematics education. The coalition goal is "not the creation of a single software product, but an understanding of how 'integration teams' comprised of developers, authors, teachers, web facilitators, and others could compose lessons by combining graphs, tables, simulations, algebra systems, notebooks and other tools available from a shared library of reusable components. Our integration teams draw upon Java versions of such powerful tools as SimCalc MathWorlds, Geometer's Sketchpad® and AgentSheets to begin addressing the needs of five new middle school mathematics curricula for empowering web-based learning technology." (www.escot.org/External/background.html). This is an exciting vision. Participating in a community such as this will be of great benefit to those who wish to either create or use reusable educational software.

This dissertation also facilitates the support of a number of future plans that will provide aids to professional development for local teacher communities. They include providing an online repository for teachers to store and reuse simulations and simulation components, collaborating with a group to create components for the online repository, and studying the impact of this repository on creating a virtual community of local peers.

7.5 Conclusion

The contribution of this work is simple. We want to aid teachers in their day-to-day activities by providing a software tool that will act as a curricula development aid. The largest benefit is that teachers can greatly improve their efficiency, change the style of their presentations and modify their delivery of curricula. The fruition of this research effort will increase the accessibility of programming systems and give teachers a tool that will empower them to create, reuse, and modify educational software. Our belief is that by empowering these teacher developers we will make a step in the direction of providing

more software for teachers. While empowering teachers to have more ownership of crafting their own software materials in a simulation environment, we also hope that a beneficial side effect of this type of environment will provide more educational stimulus for their students instead of drill and practice routines. We envision teachers creating simulations for the classroom, reusing simulations created by other educational developers, teachers and students working together to create simulation, and students utilizing simulations to learn about new phenomena and helping these students to build their own intellectual roadmaps and intellectual hierarchies, thereby creating lasting frames of references for understanding science phenomena.

The SimBuilder visual programming environment has much promise in that it has proved useful as a simulation construction kit, and educators were able to create basic educational simulations using it. SimBuilder makes use of visual programming with a direct manipulation paradigm, but allows the user of text for power users. It demonstrates programming support, while not sacrificing ease of use. It is a malleable tool that can be modified by the teacher developer to create simulations to meet their content needs, and is not restricted to one content area or predefined scenes. With its robust drawing facility the teacher developers can create visions that appear video game-like and peak the interest and curiosity of other teachers and students. In support of our prior works, it extends our study of minimalism and proves to be an effective tool for training novice programmer simulation developers.

When a teacher has the ability to realize a pedagogical need for technology, craft a tool to fill that need, and immerse it within their curriculum, he or she will be able to provide more valuable input, and have broader control over the content and creation of educational technology. The main methods of instruction employed by teachers and the tools they utilize on a daily basis have changed very little in decades. With new tools that meet their educational needs, teachers can begin to revolutionize education.

REFERENCES

- Anderson, C.L., Knussen, & Kibby, M.R. (1993). Teaching teachers to use HyperCard: A minimal manual approach. *British Journal of Educational Technology*, 24(2)
- Arlitt, M., & Williamson, C. (1997). Internet web servers: Workload characterization and performance implications. *IEEE/ACM Transactions on Networking*.
- Baroth, E. and Hartsough, C. (1995). Visual programming in the real world. In Burnett, M. M., Goldberg, A., and Lewis, T. G. (Eds.), *Visual Object-Oriented Programming*. Prentice-Hall.
- Bias, R. (1991). Walkthroughs: Efficient collaborative testing. *IEEE Software*, 8(5), 94-95.
- Blattner, M., Sumikawa, D., & Greenberg, R. (1989). Earcons and icons: Their structure and common design principles. *Human-Computer Interaction*, 3(1), 11-44.
- Borning, A. (1981). The programming language aspects of ThingLab, a constraint-oriented simulation laboratory. *ACM Transactions on Programming Languages and Systems*, 3(4), 353-387.
- Brand, C., Radar, C., Carlone, H., & Lewis, C. (1998). *Prospects and challenges for children creating science models*. San Diego, CA: National Academy for Research in Science Teaching.
- Brooks, F. (1995). *Mythical man month*. New York: Addison Wesley Longman, Inc.
- Brownstone, L., et al. 1985 Programming expert systems in OPS5: An introduction to rule based programming. Reading, MA: Addison Wesley.
- Bruce, B. C. & Easley, J. A., Jr. (2000). Emerging communities of practice: Collaboration and communication in action research. In *Educational Action Research*, v8, 243-259. November 2.
- Bruner, J.S. (1960). *The process of education*. Cambridge, MA: MIT Press.
- Burnett, M.M., & Ambler, A.L. (1992). A declarative approach to event-handling in visual programming languages. *IEEE Workshop on Visual Languages*, Seattle, 34-40.
- Budd, T. (2002). *An introduction to object oriented programming*. Boston, MA: Addison Wesley.
- Carroll, J.M. (1990). *The Nurnberg funnel: Designing minimalist instruction for practical computer skill*. Cambridge, MA: MIT Press.

- Carroll, J.M. (1995). *Scenario-based design*. New York: Wiley.
- Carroll, J. M., & Rosson, M. B.(1995). Managing and evaluation of goals for training. *Communications of the ACM*, Vol. 38, No. 7, pp. 40-48.
- Carroll, J.M. & Rosson, M.B. (1987). Paradox of the active user. In J.M. Carroll (Ed.), *Interfacing thought: Cognitive aspects of human-computer interaction*, (pp. 80-111). Cambridge, MA: MIT Press.
- Carroll, J. M., and Rosson, M. B. (1991). Deliberated evolution: Stalking the view matcher in design space. *Human-Computer Interaction*, Vol. 6, pp., 281-318. Lawrence Erlbaum Associates, Inc.
- Carroll, J. M., & Rosson, M. B. (1992). Getting around the task-artifact cycle: How to make claims and design by scenario. *ACM Transactions on Information Systems*, 10(2), 181-212.
- Carroll, J.M., & Rosson, M.B. (1995). Managing evaluation goals for training. *Communications of the ACM*, 38(7), 40-48.
- Carroll, J.M. & Rosson, M.B. (1996). Scaffolded examples for learning object-oriented design. *Communications of the ACM*, 39(4), 46-47.
- Carroll, J. M., & Rosson, M. B. 1996. Developing the Blacksburg electronic village. *Communications of the ACM*, 39(12), December, 69-74.
- Carroll, J. M. (2000). *Making use: Scenario-based design of human-computer interactions*. Cambridge, MA: The MIT Press.
- Cherry, G., Ioannidou, A., Rader, C., Brand, B., & Repenning, A. (1999). Simulations for lifelong learning. National Educational Computing Conference, NECC '99, Atlantic City, NJ.
- Chin, G. Jr., Rosson, M.B., & Carroll, J.M. (1997). Participatory analysis: Shared development requirements from scenarios, In S. Pemberton (Ed.), *Proceedings of Human Factors in Computing Systems* (pp. 162-197). New York: ACM Press.
- Chin, G. Jr. & Rosson, M.B. (1998). Progressive design: Staged evolution of scenarios in the design of a collaborative science learning environment. *Proceedings of Human Factors in Computing Systems CHI 98*. Los Angeles, CA, 611-618, ACM Press: NY.
- Connors, L. L. (2004, May 6). US schools need more tech savvy. *The Christian Science Monitor*. <http://www.csmonitor.com/2004/0506/p12s01-legn.html>.

- Cox, W. C. (1987) A case for NPOSS in real-time applications. *I&CS Magazine*, November.
- Cox, P. T., Giles, F. R., & Pietrzykowski, T. (1994). Prograph. In M. M. Burnett, A. Goldberg, and T. G. Lewis (Eds.) *Visual object-oriented programming* (pp. 45-66). Greenwich, CT: Manning.
- Cypher, A. & Smith, D. C. (1995). KIDSIM: End user programming of simulations. <http://www.acm.org/sigchi/chi95/Electronic/documnts/papers/ac1bdy.htm>
- Cypher, A., Halbert, D.C. Kurlander, D, Lieberman, H., Maulsby, D., Myers, B.A., & Turransky, A. (Eds.). (1993). *Watch what I do: Programming by demonstration*. Cambridge, MA: MIT Press.
- Dunlap, D, Neale, D., & Carroll, J. (2000). Teacher collaboration in a networked community. *Educational Technology & Society*, 2(3), p. 442-454.
- Dykstra-Erickson, E., Mackay, W., & Arnowitz, J.(2001). *Design (of) Interactions* March + April 2001.
- Edel, M. (1986). The Tinkertoy graphical programming environment. *IEEE Proceeding COMPSAC*, pp. 466-471.
- Finzer, W., & Gould, L. (1984, June). Programming by Rehearsal: An environment for developing educational software. *BYTE*, 9(6), 187-210.
- Gilmore, D. D., Pheasey, K., Underwood, J., & Underwood, G. (1995). *Learning graphical programming: An evaluation of KidSim*. ESRC Centre for Learning Research Psychology Dept. University of Nottingham, UK.
- Glinert, E.P. (1984). Pict: An interactive graphical programming environment. *IEEE Computer*, pp. 7-25. Reprinted in *Visual programming environments* (pp. 265-283). Computer Society Press.
- Glinert, E.P. (1990a). *Visual programming environments: Applications and issues*. IEEE Computer Society Press Tutorial, Computer Society Press.
- Glinert, E.P. (1990b). *Visual programming environments: Paradigms and systems*. IEEE Computer Society Press Tutorial, Computer Society Press.
- Galbraith, P., Renshaw, P., Goos, M. & Geiger, V. (1999). Technology, mathematics, and people: Interactions in a community of practice. In J. M. Truran & K. M. Truran (Eds), *Proceedings of the Twenty-second Annual Conference of the Mathematics Education Research Group of Australasia* (pp. 223–230). Adelaide: Mathematics Education Research Group of Australasia.

- Gould, L., & Finzer, W. (1984). *Programming by Rehearsal*. Technical Document. Xerox PARC.
- Gould, J. D., & Lewis, C. H. (1985). Designing for usability: Key principles and what designers think, *Communications of the ACM*, 28(3), 300-311.
- Guzdial, M. (1999). Squeak: Object-oriented design with multimedia applications. GVU Center and EduTech Institute. Draft, Dec. 1999.
- Hickok, E. (1998). Higher standards for teacher training. *Policy Review* 91(2), <http://www.policyreview.org/sept98/labs.html>
- Isenhour, P., Rosson, M.B & Carroll, J.M. (1999). Supporting interactive collaboration on the Web with CORK. *Interacting with Computers*, special issue on *Interfaces for the Active Web*.
- Ives, B., & Olson, M. H. (1984). User involvement and MIS success: A review of research. *Management Science*, 30 (5), 586-603.
- Jackiw, N. (1992). Geometer's Sketchpad.
- Johnson, J. (1991). Effect of modes and mode feedback on performance in a simple computer task, *HP Labs Technical Report HPL-91-167*.
- Johnson, J. (2000). *GUI bloopers: Don't and do's for software developers and Web designers*. San Francisco: Morgan-Kaufmann.
- Kay, A. (1993). *The Early History of Smalltalk*. HOPL-II/4/93/MA, USA. gagne.homedns.org/~tgagne/contrib/EarlyHistoryST.html
www.iam.unibe.ch/~ducasse/WebPages/FreeBooks/SmalltalkHistoryHOPL.pdf
- Kirsch, R. A. (1964). Computer interpretation of English and text and picture patterns. *IEEE Transactions on Electronic Computers*, 13, 363-376.
- Koenemann, J., Carroll, J.M., Shaffer, C.A., Rosson, M.B., & Abrams, M. (1998). Designing collaborative applications for classroom use: The LiNC project. In A. Druin (Ed.), *The design of children's technology*. San Francisco: Morgan-Kaufmann.
- Kuyper, M. (1998). *Knowledge engineering for usability: Model-mediated interaction design of authoring instructional simulations*. Unpublished dissertation, University of Amsterdam, The Netherlands.

- Kyza, E. A., Golan, R., Reiser, B. J., & Edelson, D. C. (2002). Reflective inquiry: Enabling group self-regulation in inquiry-based science using the Progress Portfolio tool. Northwestern University, 2115 North Campus Drive, Evanston, IL 60208.
- Landay, J. A. (1996). *Interactive sketching for the early stages of user interface design*. Ph.D. Dissertation, Carnegie Mellon University, Computer Science, Pittsburgh - USA.
- Levin, B.B. (2003). *Case studies of teacher development: An in-depth look at how thinking about pedagogy develops over time*. Mahwah, NJ: Lawrence Erlbaum.
- Lewis, C. (1982). Using the “thinking-aloud” method in cognitive interface design. Technical Report RC9265, Yorktown Heights, NY: Watson Research Center.
- Lewis, C., & Wharton, C. (1997). Cognitive walkthroughs. In Helander, M. G., Landauer, T. K., & Prabhu, P. V. (Eds.), *Handbook of human-computer interaction*, 717-732. 2d ed. Amsterdam, The Netherlands: North-Holland.
- Lewis, C., Brand, C., Cherry, G., & Rader, C. (1998, April). Adapting user interface design methods to the design of educational activities. *Proceedings of Human Factors in Computing Systems*
- Lieberman, H. (Ed.). (2001). *Your wish is my command: Programming by example*. San Francisco, CA: Morgan Kaufmann.
- Lieberman, H. (1993). Tinker: A programming by demonstration system for beginning programmers. In A. Cypher et al. (Eds.). *Watch what I do: Programming by demonstration* (pp. 48-64). Cambridge, MA: MIT Press.
- Lortie, D. (1975). *Schoolteacher: A sociological perspective*. Chicago: University of Chicago Press.
- Van der Meij, H., & Carroll, J. M. (1995). “Principles and heuristics for designing minimalist instruction. *Technical communication* 42:243-262
- Morimoto, K., Gregory, J., & Butler, P. (1973). Notes on the context for learning. *Harvard Educational Review*, 43(2), 263-280.
- Myers, B. (1987). Creating user interfaces by demonstration: The Peridot user interface management system, *SIGGRAPH '87 Video Review*, Vol. 59, No. 2, ACM.
- Myers, B. (1990). Taxonomies of visual programming and program visualization, *Journal of Visual Languages and Computing*, Vol. 1, No. 1, March, pp. 97 - 123

- Myers, B. (1993). Why are human-computer interfaces difficult to design and implement? *Technical paper CMU-CS-93-183*, Carnegie Mellon University.
- Myers, B. (1998, April). Scripting graphical applications by demonstration. *ACM transactions of Human Factors in Computing Systems*, 251-256.
- Myers, B.A., & McDaniel, R. (2001). Demonstrational interfaces: Sometimes you need a little intelligence, sometimes you need a lot. In H. Lieberman (Ed.), *Your wish is my command: Programming by example* (pp. 45-60). San Francisco: Morgan Kaufmann.
- National Instruments Corp. (2001). *LabVIEW Manual*, 1987. LabVIEW Demo Package.
- Nielsen, J. (1994a). Heuristic evaluation. In Nielsen, J., & Mack, R. L. (Eds.), *Usability inspection methods*, 25-62. New York: John Wiley & Sons.
- Nielsen, J. (1994b). Guerrilla HCI: Using discount usability engineering to penetrate the intimidation barrier. In R. G. Bias & D. J. Mayhew (Eds.), *Cost-justifying usability* (pp. 245-272). Boston, MA: Academic Press.
http://www.useit.com/papers/guerrilla_hci.html.
- Norman, D. A. (1986). Cognitive engineering. In Norman, D. A., & Draper, S. W. (Eds.), *User centered system design: New perspectives on human-computer interaction* (pp. 32-65). Hillsdale, NJ: Lawrence Erlbaum Associates.
- Papert, S. (1977). A learning environment for children. R. J. Seidel, & M. Rubin (Eds.), *Computers and communication: Implications for education* (pp. 271-278). New York: Academic Press.
- Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books.
- Patton, M.Q. (1978). *Utilization-focused evaluation*. Beverly Hills, CA: Sage.
- Patton, M.Q. (1981). *Creative evaluation*. Beverly Hills, CA: Sage.
- Patton, M.Q. (1982). *Practical evaluation*. Beverly Hills, CA: Sage.
- Patton, M.Q. (1990). *Qualitative evaluation and research methods*. 2nd Edn. London: Sage.
- Perrone, C., & Repenning, A. (1998). Graphical rewrite rule analogies: Avoiding the inherit or copy & paste reuse dilemma. *Proceedings of the IEEE Symposium of Visual Languages*, pp. 40-46. Nova Scotia, Canada: Computer Society.

- Polson, P., Lewis, C., Rieman, J., & Wharton, C. (1992). Cognitive walkthroughs: A method for theory-based evaluation of user interfaces. *International Journal of Man-Machine Studies*, 36, 5, 741-773.
- Potter, R. (1999). Rehearsal World.
<http://www.cs.umd.edu/hcil/pda/thesis/pda/node25.html>. Sun Apr 18 14:22:03 EDT 1999
- Powell, G.C., & Okey, J.T. (1994). Performance support for multimedia authoring. *Performance and Instruction*, 33(4), 27-32.
- Provenzo, E.F. Jr., Brett, A., & McCloskey, G.N. (1999). *Computers, curriculum and cultural change: An introduction for teachers*. Mahwah, NJ: Lawrence Erlbaum.
- Rader, C., Brand, C., & Lewis, C. (1997). Degrees of comprehension: Children's understanding of a visual programming environment. In *Proceedings of the 1997 Conference of Human Factors in Computing Systems*, 351-358. Atlanta, GA: ACM Press.
- Rader, C., Cherry, G., Brand, C., Repenning, A., & Lewis, C. (1998). Principles to scaffold mixed textual and iconic end-user programming languages. In *Proceedings of the 1998 IEEE Symposium of Visual Languages*, (pp. 187-194). Nova Scotia, Canada: Computer Society.
- Read, Jeff. (2002). Squeak if you love GUI, *World Tech Tribune*, September 6, 2002.
http://www.squeakland.org/school/HTML/essays/love_gui.html
- Repenning, A. (1993). AgentSheets: A tool for building domain-oriented dynamic, visual environments. Unpublished Masters thesis, University of Colorado Boulder.
- Repenning, A., (1994). Bending icons: Syntactic and semantic transformation of icons. *Proceedings of the IEEE Symposium on Visual Languages* (pp. 296-303). St. Louis, MO: IEEE Computer.
- Repenning, A. (1995). Bending the rules: Steps toward semantically enriched graphical rewrite rules. *Proceedings of Visual Languages* (pp. 226-233). Darmstadt, Germany: IEEE Computer Society.
- Repenning, A. (2000). AgentSheets®: An interactive simulation environment with end-user programmable agents. *Interaction 2000*. Tokyo, Japan.
- Repenning, A., & Ambach, J. (1996). Tactile programming: A unified manipulation paradigm supporting program comprehension, composition and sharing, *Proceedings of the 1996 IEEE Symposium of Visual Languages*, Boulder, CO, Computer Society, pp. 102-109.

- Repenning, A., & Ambach, J. (1997). The Agentsheets behavior exchange: Supporting social behavior processing, *CHI 97, Conference on Human Factors in Computing Systems*, Extended Abstracts(pp. 26-27). Atlanta, GA: ACM Press.
- Repenning, A., & Sumner, T. (1995). Agentsheets: A medium for creating domain-oriented visual languages, *IEEE Computer*, Vol. 28, pp. 17-25.
- Repenning, A., Rausch, M., Phillips, J., & Ioannidou, A. (1998). Using agents as a currency of exchange between end-users, *Proceedings of the WebNET 98 World Conference of the WWW, Internet, and Intranet*, Orlando, FL, Association for the Advancement of Computing in Education, 762-767.
- Roschelle, J., DiGiano, C., Koutlis, M., Repenning, A., Phillips, J., Jakiw, N., et al. (1999, September). Web based learning and collaboration: Developing educational software components. *Computer Innovative Technology for Computer Professionals*, 50-58.
- Rosson, M.B., & Alpert, S. (1990). The cognitive consequences of object-oriented design. *Human-Computer Interaction*, 5, 345-379.
- Rosson, M. B., & Carroll, J. M. (1993). Active programming strategies in reuse. In *Proceedings of ECOOP '93—Object-Oriented Programming*. Springer-Verlag, Berlin, 4–18.
- Rosson, M. B., & Carroll, J. M. (1995). Narrowing the specification-implementation gap. In J. M. Carroll (Ed.) *Scenario-based design: Envisioning work and technology in system development*, pp. 247-277. New York: Wiley.
- Rosson, M.B., & Carroll, J.M. (1996a, April). Scaffolded examples for learning object-oriented design. *Communications of the ACM*, 39(4).
- Rosson, M.B., & Carroll, J.M. (1996b, September). The reuse of uses in Smalltalk programming. *ACM Transactions on Computer-Human Interaction*, 3(3), pp. 219-253.
- Rosson, M. B., & Carroll, J. M. (2002). *Usability engineering: Scenario-based development of human computer interaction*. San Francisco, CA: Morgan Kaufmann.
- Rosson, M.B., Carroll, J.M., & Bellamy, R.K.E. (1990, April). Smalltalk scaffolding: A case study of minimalist instruction. *CHI 1990 Proceedings*, 423-429.
- Rosson, M.B., Carroll, J.M., Seals, C.D., & Lewis, T.L. (2002, July). Community design of community simulations. In *Proceedings of Designing Interactive Systems*. London: ACM Press.

- Schlager, M., Fusco, J., & Schank, P. (1998). *Cornerstones of an on-line community of education professionals*. Menlo Park, CA: Stanford Research Institute.
- Schmucker, K. (1999, Spring). A taxonomy of simulation software: A work in progress. *Learning Technology Review*, 40-75.
- Scriven, M. (1967). The methodology of evaluation. In R. Tyler, R. Gagne & M. Scriven (Eds.), *Perspectives of curriculum evaluation* (pp. 39-83). Chicago: Rand McNally.
- Seals, C., Rosson, M., Carroll, J., Lewis, T., and Colson, L. (2002). Fun learning Stagecast creator: an exercise in minimalism and collaboration, in *Proc. IEEE Symp. on Human-Centric Computing 2002* (Arlington VA, Sept.), IEEE, 177-186.
- Sebesta, R.W. (1999). *Concepts of programming languages*. 4th Edn. Reading, MA: Addison Wesley Longman.
- Shneiderman, B. (1983). Direct manipulation: A step beyond programming languages. *IEEE Computer*, 16(8), 57-69.
- Shneiderman, B. (1998). *Designing the user interface: Strategies for effective human-computer interaction*. 3rd edition. Reading, MA: Addison Wesley.
- Shu, N. (1988). *Visual programming*. New York: Van Nostrand Reinhold.
- Shu, N., (1985, August). Visual programming languages: A perspective and a dimensional analysis. *International Symposium on New Directions in Computing*. pp. 326-334. Trondheim, Norway.
- Smith, D. C., Cypher, A., & Spohrer, J. (1994). KidSim: Programming agents without a programming language. *Communications of the ACM*, 37(7), 54-68.
- Smith, D.C. (1993). Pygmalion: An executable electronic blackboard. In A. Cypher et al. (Eds.). *Watch what I do: Programming by demonstration* (pp. 19-47). Cambridge, MA: MIT Press.
- Soloway, E. (1998). Log on education: No one is making money in educational software. *Communications of the ACM*, 41(2), 11-15.
- Soloway, E., Guzdial, M., & Hay, K. (1994). Learner-centered design: The challenge for HCI in the 21st century, *HCI interactions*, 36-48.
- Stevens, A. (1992, December). A conversation. *Dr. Dobb's Journal*, 195, 7-14.
- Stevens, J. (1978). *Applied Multivariate Statistics for Social Sciences*. 4th Ed., Mahwah, N. J.: Erlbaum. Electronic Reproduction. Boulder, CO. NetLibrary 2001, p699.

- Tewari, R. & Gitlin, D. (1994). On object-oriented libraries in the undergraduate curriculum: Importance and effectiveness. *Proceedings of the ACM SIGCSE*, 26(1).
- Troen, V, & Boles, K. (2003). *Who's teaching your children?: Why the teacher crisis is worse than you think and what can be done about it*. New Haven & London: Yale University Press.
- Tyack, D. & Cuban, L. (1995). *Tinkering toward utopia: A century of public school reform*. Cambridge, MA: Harvard University Press.
- Ungar, D., Smith, R., & Self (1987) The power of simplicity. In *Proc. OOPSLA '87, Object Oriented Programming Systems, Languages and Applications*, pages 227-241, 1987. Also published in *Lisp and Symbolic Computation* 4(3), Kluwer Academic Publishers, June, 1991.
- van Deursen, K., Klint, P. & Visser, J. (2000). Domain-specific languages: An annotated bibliography. Research paper: www.telin.nl/dscgi/ds.py/Get/File-11929/dslbib.pdf.
- Vygotsky, L. (1978). *Mind in society*. Cambridge, MA: Harvard University Press. www.tue.nl/ipo/people/krahmer/abc-allwood.ps.
- Wharton, C., Rieman J. Lewis, C., & Polson, P. (1994). The cognitive walkthrough method: A practitioner's guide. In J. Nielsen, & R. Mack (Eds.), *Usability inspection methods*. New York: John Wiley & Sons.
- Wissman, J. (2002). Minimalist and traditional training methods for older adults: A comparative study in a software environment. Unpublished Masters thesis. Virginia Tech, Blacksburg, VA.
- Woodfield, S., Embley, D., & Scott, D. (1987). Can programmers reuse software? *IEEE Software*, July, 52-59.

APPENDICES

Appendix A: Taxonomy of Visual End User Programming Environments for Educational Simulations	273
Appendix B: Evaluation Matrix of 5 Environments	276
Appendix C: Evaluation of Stagecast	278
Appendix D: Institutional Review Board Forms: IRB Approval	280
Appendix E: Institutional Review Board Forms: Informed Consent.....	282
Appendix F: Pre-Questionnaire	286
Appendix G: AgentSheets Learning and Reuse Tutorials.....	291
Appendix H: SimBuilder Learning and Reuse Tutorials	308
Appendix I: Post-Questionnaire.....	328
Appendix J: Post-Questionnaire Data and Statistical Analysis	334
Appendix K: Retrospective Interview Questions	351
Appendix L: AgentSheets Simulation Creation Timing, Objects and Rules Created	353
Appendix M: SimBuilder Simulation Creation Timing, Objects and Rules Created	362
Appendix N: SimBuilder Session Transcripts	365
Appendix O: AgentSheets SimBuilder Artifact Comparison.....	386
Vitae	390

Appendix A: Taxonomy of Visual End User Programming Environments for Educational Simulations

Techniques

DM -Direct Manipulation

Text - Text Programming

VP -Visual Programming

Kit -Components/Patterns/Construction Kit

Sheet - Spreadsheet

System and Author(s)	Date	Description	Techniques	Applications	Major Contributions
ActivChemistry	1998	Educational simulation chemistry construction kit. It provides user a fixed set of parts they can combine to perform experiments. Designed to teach new concepts.	Kit	Educational Files are small/easily shared. Simulations are typically smaller than 20kilobytes and are therefore very easy to deliver over networks. Uses <i>chemistry set metaphor</i>	Presents detailed scientific concepts in a direct manner. Enables students to perform expensive or dangerous experiments virtually. Provides a cost-effective way to increase lab time.
AgentSheets Alex Reppenning	1997	Simulation builder that contains a construction that users can drag rules from palettes to create behaviors for agents. Constraint based.	DM Text Kit Spread-sheet	General Domain Independent Graphical rewrite rules and Uses <i>Spreadsheet metaphor</i>	Automatically generate related icons and has a behavior processor that enables users to build programs by combining pieces. It uses programming by analogy as its concept of behavior reuse.
Amulet Brad Myers	1994	Amulet is a user interface development environment for C++. Successor to Garnet.	DM Text Kit	Interactive User Interfaces	Amulet includes many features specifically designed to make the creation of highly-interactive, graphical, direct manipulation user interfaces significantly easier, including a prototype-instance object model, constraints, high-level input handling including automatic undo, built-in support for animation and gesture-recognition, and a full set of widgets.
ARK Randall Smith	1987	Animated environment for creating interactive simulations.	DM Text Kit	General: labs, physical phenomena Uses <i>Physical world metaphor</i>	ARK simulations are intended to facilitate an intuitive understanding of the simulation's interactive rules by making them appear as accessible physical objects called interactors. The environment supports creation and modification.
FABRIK Ludolph	1988	Dataflow programming in circuit board metaphor.	DM VP PBD		
Garnet Brad Myers	1990	Two complete widget sets. One with a Motif look and feel implemented in Lisp, and one with a custom look and feel. Interactive design tools for creating parts of the interface without writing code.	DM Text	<ul style="list-style-type: none"> - OOP programming prototype-instance, model. - Automatic constraint maintenance. - Built-in, high-level input event handling. - Support for gesture recognition. - Widgets for multi-font, 	

				multi-line, mouse-driven text editing. Optional automatic layout of application data into lists, tables, trees, or graphs. – Automatic generation of PostScript for printing. – Support for large-scale applications and data visualization.	
Geometer's Sketchpad	1992	Construction kit for simulations of geometric theorems. Allows a student to discover and explore math.	DM PBD	Geometric Uses <i>Sketching Metaphor</i>	Drawings can be manipulated by the user, but will still maintain their geometric "knowledge". Expandable user construction set and dynamic updating of constraints.
HI-VISUAL Hirakawa	1986	Extensible icon general programming	DM Text VP	Visual Interactive Programming	
KidSim(Cocoa) Cypher, A. and Smith, D. C. Clayton Lewis	1997	Simulation building environment. System allows user to program by DM. Draws on four traditions: production systems, graphical rewrite rules, PBD and simulations.	DM PBD VP Text Spreadsheet	General – Domain Independent	The combination of PBD and graphical rewrite rules is stronger than either. "Graphical rewrite rules solve PBD representation problem, and PBD solves the rule-semantics problem."
LabVIEW by National Instruments	1998	LabVIEW is a scientist's construction kit to build Virtual Instruments by direct manipulation. Within the system they have three palettes, which are used for editing, debugging and creating. Tools Palette, Controls Palette and Functions Palette. There are Wizards and a Tutorial included.	DM Text Kit	Virtual Laboratory Instruments	National Instruments graphical software, LabVIEW, speeds the work of engineers and scientists because they can quickly program their instruments to meet their specific needs. Interactive graphic for Test and measurement, data acquisition and control, laboratory automation, Process monitoring and control.
Model-It		Mathematical based simulation generator. User-friendly interface.		Domain Independent	Allows user to enter equations in pop-up menus with English like syntax.
Mondrian	1992	An inscrutable graphical editor. User demonstrates graphical edits and the system infers constraints. Can reuse commands.	PBD	Uses <i>Teaching Metaphor</i>	Represents all operations as storyboards with examples. Procedural. User can generalize actions to define new operations
NoPumpG Clayton Lewis	1990	Proposes that peoples success with spreadsheets implies that a large class of solutions in that model.	DM VP Spreadsheet	Uses Spreadsheet Metaphor	Additions to spreadsheet model for interactive graphics
PICT Ephriam Glinert	1984	Iconic flowchart programming. Fully interactive iconic flowchart programming system designed for novice programmers	Text VP Kit	Iconic Flowchart programming	
Pinball Construction Set		Simulation of creating a pinball game. Using the provided components to build.	DM Kit	Pinball Game Construction	Entertaining, but limited to preprogrammed behavior.
PROGRAPH PROGRAPH2	1988	VPL used data flow and class hierarchies that are entirely pictorial	DM VP	General	Variables and logic syntax are replaced by graphical connections
Pygmalion David Canfield DCSmith	1975	1 st PBD system and gives interactive feedback. User interaction technique is simply choosing operations and	VP PBD Kit	Uses Blackboard Metaphor	At the time was a big improvement over traditional programming, because the user programs by operating

		opcodes from a menu and specializing them once they are placed on the blackboard.			on actual values, which are interpreted as changes are made. It allowed user ideas to be sketched out on the blackboard and these icons could then be utilized to represent or control programs
Rehearsal World Gould & Finzer	1984	Programming by Rehearsal. Emphasis of visual. Only things that can be seen can be manipulated.	DM Kit VP PDB	Educational Software Uses <i>Programming by Rehearsal Metaphor</i>	Extends traditional programming by adding a macro recorder. It replaces writing lines of code with demonstrations, avoiding complications of syntax.
Sketchpad Ivan Sutherland	1965	Ivan Sutherland's landmark program. 1 st graphical system.			
SimCalc Kaput	1994	Combines advanced simulation technology with innovative curriculum that begins in the early grades and includes powerful ideas extending beyond classical calculus	DM VP	Mathematics	
SimCity SimCity3000	1999	Simulation of city building. Using the provided components to build.	DM Kit	City Planning	3000 has improved graphics and example cities.
SmallStar Daniel Halbert	1984	Simulation of Xerox Star with PDB added.	VP PDB		
Squeak Disney Imagineering	1996	Development environment in which to build educational software by non technical people and children	Kit DM VP PBD Text	Educational, 2D and 3D color graphics, multivoice sampled and synthesized sounds, animations, video, midi flash, jpeg, gif	Practical Smalltalk where user can examine source code of every part of the system, including graphics primitives and the VM. All in Smalltalk
Stagecast Stagecast Software	1998	Simulation building environment. System allows user to program by DM	VP DM Text PDB	General	Based on Cocoa, but build with Java to be platform independent.
Star Logo MIT Media Lab From Papert's LOGO	1997	StarLogo is a programmable modeling environment for exploring the workings of decentralized systems.	Text	Real life phenomena	Procedural
Star Wars Droid Works Lucas Learning	1998	Simulation environment to build computer droids to meet given specifications. Must apply principles from math, physics and biology.	VP Kit	Educational	Adds a gamelike quality to keep users attention. Setting is an immersive environment.
ThingLab Alan Borning	1981	OO constraint programming system for modeling physical systems	DM Text VP	Geometric and Physical Systems	Objects modified by changing constraints that determine their shape and behavior.
Tinker Lieberman	1980	Example oriented environment for novice programmers. Users program by providing concrete examples.	Text PBD	Programming Uses <i>Teaching Metaphor</i>	Users program by providing concrete examples.
Tinkertoy Mark Edel	1986	Build lisp programs by connecting icons with 2 inputs and one output	DM Text VP	General- will support any program that can be created in LISP	Programs built with icons not text
Toon Talk Ken Kahn	1999	Immersive programming environment where the user learns programming fundamentals.	DM Kit Text	Programming	Present all programming concepts as concrete entities in system. Procedural.
VEE					
Xerox Star		Beginning of GUI desktop	DM VP		

Appendix B: Evaluation Matrix of 5 Environments

	Environments Studied				
	<i>AgentSheets</i>	<i>Cocoa</i>	<i>Hyperstudio</i>	<i>Stagecast</i>	<i>Squeak/SimBuilder</i>
<i>Environment Issues</i>					
Direct Manipulation	3	2	5	2	1
Rule based to contain more functionality	2	3	5	3	2
Support realistic actions (non-determinism)	3	4	5	4	1
Continuous motion	4	4	5	4	1
Minimal number of windows	4	2	1	2	1
Ease of use for novices	4	2	1	3	1
Standard basic set of components (e.g. condition/actions)	2	5	1	5	2
Library of simulations organized by different Classes	5	5	3	5	3
Object Template (Standard basic set of reusable functionality Objects)	5	5	3	5	1
New Worksheet based on template	3	3	3	3	
Usable on any platform	4	4	4	2	1
Easily portable to web applications (e.g. JAVA)	1	1	1	1	1
Can run Sims at varying speeds	4	1	5	1	1
Easy to reuse an object	5	1	1	1	1
Easy to erase an object	4	2	1	3	1
Easy to copy an object	2	1	1	2	1
Easy save that prompts user if not already saved	1	1	1	1	1
Easily change name of object	5	1	1	1	1
Easily understood icons	3	3	3	3	1
Incremental testing	2	2	5	2	1
Good Menu layout	4	2		2	1
Special functions: Change Sim state	3	4	5	4	
Easy to open environment	4	2	2	2	1
Supports the opening of two projects simultaneously	5	5	?	1	1
Support classes of similar objects	4	5	4	1	

	Environments Studied				
	<i>AgentSheets</i>	<i>Cocoa</i>	<i>Hyperstudio</i>	<i>Stagecast</i>	<i>Squeak/ SimBuilder</i>
<i>Drawing Tools</i>					
Standard toolkit: Square, circle, triangle	5	1	1	1	1
Easy clear	5	1	1	1	1
Undo	5	3	1	3	1
Paint Fill (e.g. paint bucket)	3	3	1	3	1
Graduated Block Fill	2	5	5	5	1
Choose drawing point size	5	1	1	1	1
Choose erase point size	5	1	1	1	1
Import objects (e.g. from file/from clip art gallery)	4	1	1	1	1
Color Matcher from any color on desktop	4	4	5	4	1
Variable size objects within worksheets	5	5	5	2	1
<i>Rules</i>					
Graphical Rewrite Rules	2	5	5	5	NA
Power User mode (like move up, the user just does it and the system creates the corresponding rule)	3	4	5	3	
Easily understood rule ordering	4	4	5	3	NA
See rule actually fire with indicator light/highlight	3	5	5	1	
Appropriate sound to indicate operation	3	5	5	5	1

Ranking Key

1. Satisfies Requirements
2. Above minimum requirements
3. Meets minimum requirements
4. Below requirements
5. Failed to meet minimum requirements

Appendix C: Evaluation of Stagecast

Visual Programming Languages Group

Stage Cast Evaluation of Trail Version April 13, 1999

Pros

- Can have different size characters on the same stage.
- It will run many speed variations. Four are available marked from turtle to rabbit speed.
- Clock can be tied to real time clock. Run of program makes clock cycles more apparent to the user.
- Can enter text comments on rules with quote tool.
- There is an area where you can put like objects called Jars where you can store instantiations of a class. If you have a fish class
- Erase will erase all characters by just a click and hold mouse down.
- Tutorial is really great and animated.
- ? is a good icon and the help area is clear.
- Can open multiple stages, with their own characters. Stages can share characters
- Can have multiple worlds open at once. Can drag characters from one world to the next.
- Can delete stages from the stage palette with either vacuum or delete key.
- Can Change depiction easily with just a right click and select the new depiction.
- Teleportation door is a nice feature. Characters can jump from one location to another on the stage or can teleport to another stage.
- **Jars to store classes.** We could have many different fish with similar behavior. A shark could eat all classes of fish.
- **Time out.** Area where character is still visible but does not function.
- Icon for loading application.
- Function Palette very clear with easy access to many sections for the following:
characters, special, stages, sounds, jars, and global.
- Can use rule buster function "like ghost buster" to skip evaluation of rules during execution.
- Can add text heading as a title for stage.
- Stage is four sections bigger than what is visible.

Cons

- Much of the interface is initially hidden with expand button "just one very small orange button" to open parts of the interface. The button should be regular size and labeled so that it will be easy for user's to find.
- We wanted to resize screen, but no resize handles. It will resize but only at corners.
- Pull down menu not apparent. Icon was small and indistinguishable like Smalltalk pull down menu.
- Duplicate is not straightforward. You must select duplicate, the click object to duplicate, click one more to create one copy, click and hold mouse down to create multiple copies.
- No multiple or "group" copy e.g. shift click in many applications.

- Can't close Tutorial until it is finished, If you do it closes application.
- Stage buttons. There are buttons to indicate 1 stage or 2 stages and you drag the appropriate stage thumbnail into the indicated area. Not very clear and icons not very helpful.
- Delete button is a vacuum cleaner. A big eraser would be more helpful.
- Copy button is a rubber stamp. Not universally recognizable
- Record button is represented as 2 frames on a reel of tape. Not universal. Normally it is a block box with a red dot in the center.
- Blob is not universally recognizable for drawing palette. It almost looks like a flower. Paintbrush should be adequate to create new instance and begin drawing once you place it on the stage.
- Quote tool icon is not easily recognizable. The word text would be more helpful.
- Once you have named a rule with Quote tool, it is still necessary to click on "" to see name.
- Stage Section tab is a small gray arrow button that is barely visible.

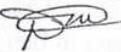
Appendix D: Institutional Review Board Forms: IRB Approval

 **Virginia Tech**
VIRGINIA POLYTECHNIC INSTITUTE
AND STATE UNIVERSITY

Institutional Review Board
Dr. David M. Moore
IRB (Human Subjects) Chair
Assistant Vice Provost for Research Compliance
CVM Phase II - Duckpond Dr., Blacksburg, VA 24061-0442
Office: 540/231-4991; FAX: 540/231-6033
e-mail: moored@vt.edu

MEMORANDUM

TO: Mary Beth Rosson Carroll CS 0106
Cheryl D. Seals CSA 0106
Justin Gortner CS 0106

FROM: David M. Moore 

DATE: June 12, 2002

SUBJECT: Expedited Approval – “Learning and Reuse in Visual Programming Environments” – IRB #02-303

This memo is regarding the above-mentioned protocol. The proposed research is eligible for expedited review according to the specifications authorized by 45 CFR 46.110 and 21 CFR 56.110. As Chair of the Virginia Tech Institutional Review Board, I have granted approval to the study for a period of 12 months, effective June 11, 2002.

Approval of your research by the IRB provides the appropriate review as required by federal and state laws regarding human subject research. It is your responsibility to report to the IRB any adverse reactions that can be attributed to this study.

To continue the project past the 12 month approval period, a continuing review application must be submitted (30) days prior to the anniversary of the original approval date and a summary of the project to date must be provided. My office will send you a reminder of this (60) days prior to the anniversary date.

cc:File

*A Land-Grant University—The Commonwealth Is Our Campus
An Equal Opportunity / Affirmative Action Institution*



Institutional Review Board

Dr. David M. Moore
 IRB (Human Subjects) Chair
 Assistant Vice Provost for Research Compliance
 CVM Phase II - Duckpond Dr., Blacksburg, VA 24061-0442
 Office: 540/231-4991; FAX: 540/231-6033
 e-mail: moored@vt.edu

April 3, 2003

MEMORANDUM

TO: Mary Beth Rosson Carroll Computer Science 0106
 Cheryl D. Seals CSA 0106

FROM: David M. Moore 

SUBJECT: "Learning and Reuse in Visual Programming Environments" – IRB #02-303

The above referenced protocol was initially reviewed and approved by the Virginia Tech IRB on June 11, 2002. DHHS Office for Human Research Protections (OHRP) regulations require that on-going projects be reviewed and re-approved within 12 months (or sooner) from the date of the original approval of the protocol.

This memo is intended to ascertain the current status of your protocol, and, if needed, to prompt you to seek re-approval. Please provide the information requested below, and return a copy of this document to the Office of Research Compliance (attn: David Moore, Mail Stop 0442) at your earliest convenience.

Regarding the above referenced protocol [check the applicable statement]:

- YES, the project has been completed.**
- NO*, it has not been completed, and will require an extension beyond the original 12 month approval.**

NOTE:

If you checked NO, then you must submit the following materials for review and re-approval by the IRB Chair:

1. A copy of the original signed approval form.
2. A brief progress report to include: how many subjects were involved to date, and a description of any unforeseen complications or events which increased the level of risk to the subjects.
3. Any planned changes in the protocol which would impact the human subjects and the level of risk.
4. A re-analysis of the risks and benefits in light of the experience gained in the project to date.
5. An estimate of the time required (in months) to complete the study.

Your re-application package should be submitted through your departmental Human Subjects Committee or designated departmental IRB reviewer. Prompt completion of the package and transmittal to the Office of Research Compliance (attn: David Moore, Mail Stop 0442) will ensure timely action by the IRB prior to the approval expiration of the original protocol. As stated in the original approval letter, the request for continuation must be received **30 days prior** to the expiration date. Failure to submit the requested materials 30 days in advance of the expiration date may result in notification of the PI to stop all Human Subjects activities until updated approval is received.

Appendix E: Institutional Review Board Forms: Informed Consent

Request for Expedited Approval of Research Involving Human Subjects

[please print or type responses below]

Investigator(s): Cheryl Seals, Justin Gortner and Mary Beth Rosson

Department(s): Computer Science Mail Code: 0106 E-mail: {cseals, jgortner, rosson}@vt.edu

Project Title: Learning and Reuse in Visual Programming Environments # of Human Subjects
40

Source of Funding Support: Departmental Research Sponsored Research (OSP
No.:)

All investigators of this project are qualified through completion of the formal training program or web-based training programs provided by the Virginia Tech Office of Research Compliance.

Note: To qualify for Expedited Approval, the research activities must: (a) present not more than minimal risk to the subjects, (b) not involve any of the special classes of subjects, except children as noted, and (c) involve only procedures listed in one or more of the following categories. The full description may be found in the Expedited Review section of the Virginia Tech “*IRB Protocol Submission Instructions Document*” or 45 CFR 46.110 (<http://ohrp.osophs.dhhs.gov/humansubjects/guidance/45cfr46.htm#46.110>)

Please mark/check the appropriate category below which qualifies the project for expedited review:

- 1. Clinical studies of drugs and medical devices when proscribed conditions are met [see item (1), page 8 of the “Instructions” document].
- 2. Collection of blood samples by finger, heel or ear stick, or venipuncture subject to proscribed limitations [see item (2), page 9 of the “Instructions” document].
- 3. Prospective collection of biological specimens for research purposes by noninvasive means. Examples: hair and nail clippings, deciduous teeth, permanent teeth, excreta and external secretions, uncannulated saliva, placenta, amniotic fluid, dental plaque, muscosal and skin cells and sputum [see item (3), page 9 of the “Instructions” document].
- 4. Collection of data through noninvasive procedures routinely employed in clinical practice, excluding procedures involving x-rays or microwaves [see item (4), page 9 of the “Instructions”].
- 5. Research involving materials (data, documents, records or specimens) that have been collected or will be collected solely for non-research purposes (such as medical treatment or diagnosis [see item (5), page 10 of the “Instructions” document].
- 6. Collection of data from voice, video, digital, or image recordings made for research purposes [see item (6), page 10 of the “Instructions” document].
- 7. Research on individual or group characteristics or behavior (including, but not limited to, research on perception, cognition, motivation, identity, language communication, cultural

beliefs or practices, social behavior), or research employing survey, interview, oral history, focus group, program evaluation, human factors evaluation, or quality assurance methodologies [see item (7), page 10 of the “Instructions” document].

	Cheryl D. Seals	
Investigator(s)	Print Name	Date

	Justin Gortner	
Investigator(s)	Print Name	Date

	Mary Beth Rosson	
Investigator(s)	Print Name	Date

Departmental Reviewer	Print Name	Date
-----------------------	------------	------

Chair, Institutional Review Board	
Date	

This project is approved for _____ months from the approval date of the IRB Chair.

Purpose of Research

This research is being conducted to examine reuse techniques in visual programming environments, which can be used by teachers and pre-service teachers to create simulations for their classrooms. The participants will use the tools we provide to learn to create instructional software and finally to create their own simulations.

Justification of Project

This study will evaluate a framework that supports the design of educational software by teachers. There is a lack of good educational software to help teachers with their day-to-day duties, want to give teachers a toolkit that they can use to create, reuse and modify their own software. We will examine our teachers in the process of creating software and from their task profiles design and create an environment that serves those needs. We are seeking to ascertain the following:

- Limitations of visual programming languages and what methods are currently employed to aid novices; can we mitigate some of those limitations
- Better reuse strategies for novice programmers
- Can this research improve methods currently employed by teachers and support construction/reuse of simulations?

Procedures

Participants are selected from middle school teachers and graduate students in education and are subject area experts, but with little programming experience. This study will take place during 2002-2003.

Participants will first fill out a brief pre-test questionnaire to give the investigators an indication of their previous programming experience. Then they will be given printed instructions, and will work in pairs to explore the tutorial material presented. Investigators will intervene in the case that the subjects become disoriented in their task. There will be two sessions and each session will last approximately 1 hour. Afterwards, each participant will fill out a short post-test questionnaire to express his or her experiences about working in the visual programming environment. All empirical sessions will take place in either 102 or 104 McBryde Hall.

Subjects will be videotaped, and their interaction with the system recorded by screen capture equipment. This will allow us to review their progress during the tutorial and carefully analyze the interactions that occurred at a later time.

During the study, participants will be asked to take part in the following research activities:

Background Questionnaire: We will ask participants a number of questions related to their general educational background and computer experience. Questions will gather years of practical teaching experience and their use of software and computers in the classroom.

Evaluation: We will conduct evaluation with the following: performance, time on task, number of errors, logging critical incidents and retrospective interviews. Each experimental should take approximately one hour.

Risks

There are no known risks to you in this research. The involvement and performance of students in the research is strictly voluntary. The activity is not a task for partial fulfillment of any current class assignment.

Benefits of this Project

This research will benefit students, educators, and developers. The benefits of the research are to develop educational software for use in today's classrooms as curricula aids. The knowledge gained about the development of end users programs in a visual programming environment will benefit students, educators and instructional software developers in future research and development efforts. Our aim is to provide

information that will be used to increase the amount of usable educational software that is available for teachers. This study will also study reuse strategies in the novice programmer population and make programming more accessible for that group.

Confidentiality/Anonymity

The results of this study will be kept strictly confidential. Your written consent is required for the researchers to release any printed or electronic data attributed directly to you as an individual to anyone other than personnel working on this project. Any screenshots or images included in presentations or publications will be used anonymously.

Informed Consent

An introduction to the survey will summarize procedures of this experiment and obtain participant consent.

I _____ consent to the terms of this experiment. Filling out a brief pre-test questionnaire to give the investigators an indication of their previous programming experience. I will be given printed instructions, and explore the tutorial material presented. Investigators will intervene in the case that the subjects become disoriented in their task. There will be two sessions and each session will last approximately 1 hour. Afterwards, I will fill out a short post-test questionnaire to express his or her experiences about working in the visual programming environment. All empirical sessions will take place in either 102 or 104 McBryde Hall. None of the materials that I provide may be used without my consent beyond the scope of the research. I expect all my responses will be held as confidential as stated in the confidentiality section.

Date: _____

Appendix F: Pre-Questionnaire Fall 2002 Visual Programming

Sim ID

Age

Gender

Major

Educational background: Please list any degrees or courses taken in the following areas.

Software Design

Instructional Technology

Computer Programming

Curricula and Instruction

Other instruction that would be helpful in design

Please list any work experience.

Do you have any teaching experience?

Yes

No

If Yes, what classes did you teach?

For approximately how many years have you been using a computer?

of years

Do you have experience using a PC(i.e. IBM, Dell, Compaq, Toshiba, etc.)or Macintosh (formerly called Apple) Computer?

PC Mac Both None

How many years of PC use.

How many years of Mac use.

On average, how many times a week do you use a computer?

0-1 2-3 4-5 6 or more

On average, how many hours do you spend on your computer per week?

0-4 5-8 9-12 more than 12

Have you used a hand held computer game?

Yes

No

How many times have you played a computer game?

0-4 5-8 9-12 more than 12

How many minutes on average did each game take?

0-5 6-10 11-20 30-60 more than 60

Have you used a palm pilot?

Yes

No

How many times have you used it?

0-4 5-8 9-12 more than 12

How many minutes on average did each use take?

0-5 6-10 11-20 30-60 more than 60

Have you ever used any drawing software?

Photo Editor Adobe Photoshop Microsoft Picture It Corel Draw

Microsoft Paint None other:

If Yes, How many times have you used any drawing software?

0-4 5-8 9-12 more than 12

Have you ever done any programming ?

Yes

No

If yes, what languages have you used..

Have you taken any computer classes?

Yes

No

If yes, list software or programs used.

Do you have any previous experience with visual programming environments? (Authorware, Director, Dreamweaver, Visual Basic, etc.)?

Yes

No

If yes, what types of software packages did you use and what kinds of projects did you design or create with these packages?

Do you use a word processor, such as Microsoft Word or Word Perfect?

Yes

No

If Yes, How many documents have you created?

0-4 5-8 9-12 more than 12

Have you used a spreadsheet program, like Microsoft Excel or Quattro Pro?

Yes

No

If Yes, How many spreadsheets have you created?

0-4 5-8 9-12 more than 12

How many times do you email or chat per week?

0-1 2-3 4-5 6 or more

How many times do you use the Internet per week?

0-1 2-3 4-5 6 or more

Do you use computers in any of your classes?

- Yes
 No

For what type of activities?

Do you consider yourself more artistic, analytical or both?

- Analytical
 Artistic
 Both

What (if any) role do you see for computer simulations in primary/secondary education? (e.g. simulation of a factory and pollution it creates).

Suppose you were going to build a computer simulation of a volcano exploding for earth science. What sorts of things do you think would be involved (i.e. what objects and what do they do)?

What kinds of real world situations are you familiar with and could imagine a simulation recreating it?

In the section below, choose the response that most accurately describes you.

1. I frequently read computer magazines or other sources of information that describe new computer technology.

- Strongly Agree Agree Neutral Disagree Strongly Disagree

2. I know how to recover deleted or lost data on a computer or PC.

- Strongly Agree Agree Neutral Disagree Strongly Disagree

3. I know what a LAN is.

- Strongly Agree Agree Neutral Disagree Strongly Disagree

4. I know what an operating system is.

Strongly Agree Agree Neutral Disagree Strongly Disagree

5. I know how to install software on a personal computer.

Strongly Agree Agree Neutral Disagree Strongly Disagree

6. I know what a database is.

Strongly Agree Agree Neutral Disagree Strongly Disagree

7. I am computer literate.

Strongly Agree Agree Neutral Disagree Strongly Disagree

8. I am good with computers.

Strongly Agree Agree Neutral Disagree Strongly Disagree

Questions 1-8 above are from Computer User Understanding Survey

Potosky & Bobko 1998

Appendix G: AgentSheets Learning and Reuse Tutorials
AgentSheets Tutorial Learning Session

Exploring Visual Programming

Agentsheets Tutorial
Implementing a Model of the Water Cycle

Human Computer Interaction
@
Virginia Polytechnic Institute and State
University

This tutorial is a draft of materials being developed as part of behavioral research underway in the Computer Science Department at Virginia Tech. It is provided on an "as-is" basis; however, we welcome comments and suggestions. Please direct any feedback to seals@csgrad.cs.vt.edu.

AgentSheets Tutorial © Virginia Tech Visual Languages Group

Exploring the Water Cycle Model

- **Double click** to open the Agentsheets program.  Agentsheets
- **Select**  to open an existing project.
A dialog will be displayed that contains all projects.
- **Select Water Cycle** Folder and **Select Open**
→ This will open your first example of a simulation.
- Now, **Run**  this model.
→ Watch the Simulation. You can slow it down to see the action more clearly.
→ What actions are taking place?
- Press  and  after a few minutes of observing the model.

Exploring The Sun

Let's investigate the agents

- To investigate the **Sun** agent's look or depiction, find it in the **Water Cycle** gallery **select it** (when you select an agent a blue line will highlight it) and **press**  (you will see its 32x32-pixel representation).
→ Use the tools to change the sun from yellow to bright orange.
- To investigate the rules that govern the agents behavior, click **press** .
This will show you the set of graphical rules for this agent. The simulation is programmed by direct manipulation and graphical rules.
→ What does the **Sun** do?
- You can change and create agent behaviors with rules.
Rules can be viewed by clicking **Tools** → **Condition Palette** or 
And **Tools** → **Action Palette** or .
 You program by simply clicking on a rule, then dragging and dropping conditions and actions into the agent's behavior.
(Take a few minutes to explore conditions and actions...)
- Examine some of the more complicated behaviors by selecting the **Cloud** and reviewing its behavior.

Changing The Behavior of Clouds

 Take a moment to review the interaction guide to gain a better understanding of the interactions between agent's behaviors

Make Clouds more active. Currently the behavior of the cloud is to move 2% of the time. Find the **IF** statement that causes this behavior. Let's make the clouds move a lot more.

- Click on the desired text box (currently showing 2%) to make your change.
- **Increase the percentage to 50%** for moving to the right.
- Click **Apply** (located on the menu at the bottom of the behavior window).
- Now, **Run** the simulation to see the affects of your changes.
- Press **Stop** and **Reset** so that you can try another change.

Make the clouds move vertically. Currently the behavior of the cloud is to move horizontally across the sky. Let's try to change the behavior of the cloud so that it will move vertically. Look at the rule to **move** agents.



- In order to change the direction that the agent moves in just click on the arrow and move it until it points down.
- **Apply** your new change.
- **Run** the simulation to see how the simulation has changed.

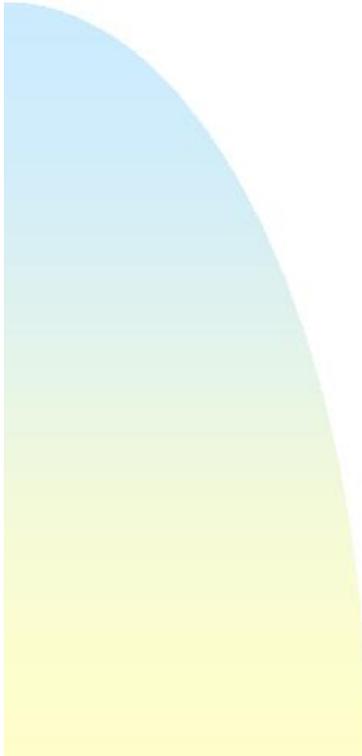
Creating a Volcano Simulation

A sample environment that you could simulate is a volcano erupting. A volcano involves the interaction of several complex factors. Pressure is built up over a period of time. Once the pressure reaches a certain level the pressure is released as sparks, smoke, lava, and heat. The lava causes the earth and the mountain to become larger as a by-product. After the volcano has erupted the pressure has been released and the Volcano becomes quiescent.

- On the next sheet Draw a simple picture of what you would expect a volcano to look like. Also identify candidate agents for your volcano simulation.

(Take 2-5 minutes brainstorming and drawing.)

Creating a Volcano Simulation



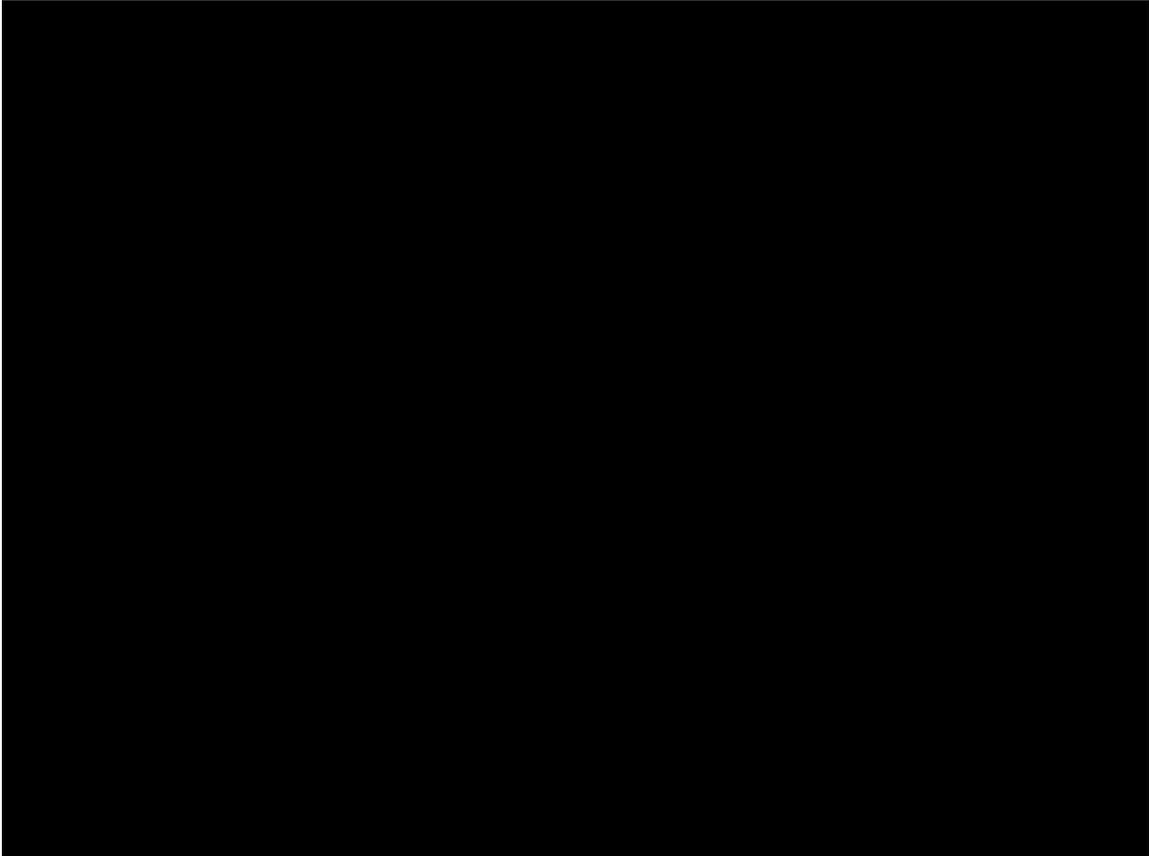
- Select  to create a **New Project**.
- Enter ***volcano_yourInitials*** in file name box.
- Specify the depiction size of the agents. For this project, we will use 32 X 32 pixels per agent.
 - This is the same size as desktop icons. A gallery will open to hold all of the agent depictions and their behaviors. You can use a crayon to paint whatever you like and the eraser to make modifications.
- *Once you have an idea of the new environment you want to create, begin by creating **new agents**.*

You can create **a mountain, sparks**(that fly out of the volcano), **lava** and any other agents that will improve the aesthetic view of your microworld. Perhaps you would like to include a sky for background, or trees, etc. If you need help creating an agent, refer to the interaction guide.

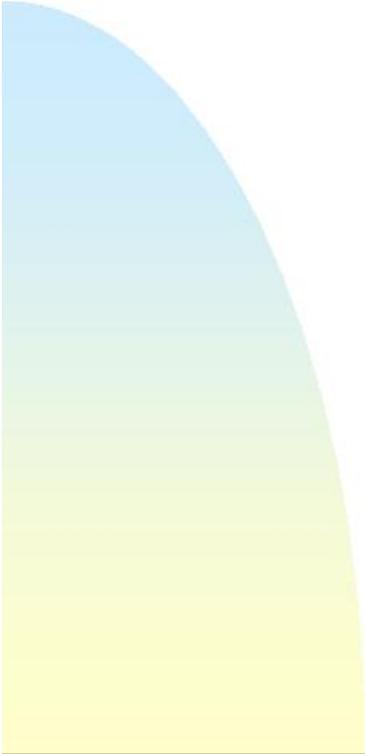
AgentSheets Tutorial2. Reuse from Starter (Component Model)

Exploring Visual Programming

Section II



Ocean World Simulation



Creating new agents and Adding behaviors to create Ocean World Simulation.

A simulation in Agentsheets is simply a set of agents that work together to create visual effects.

A key aspect of reusing a simulation is to specialize existing agents and adding new agents. We will begin with the small task of reusing a new agent from the Starter World.

Task 1. Click on Gallery

- Click on *File* → *Save*.
- Name this gallery as **Ocean_Gallery_YourInitials**

Create a new worksheet

- Click on *File* → *New Worksheet*
- Name this worksheet as **Ocean_Worksheet_YourInitials**

Task 2. Draw a new agent

 Refer to interactions guide for **Gallery/Worksheet Tools** and more help.

Task 3. Reusing and creating new behavior for your new agent

- Some agents already have behaviors. You may need to look at there behaviors to get started. You can reuse their behavior by just dragging the entire rule that you would like to reuse into your agent's behavior palette.
- Add new behaviors.

 Refer to interactions guide for more *Help*.

- **Think of other interactions to make your Ocean World simulation an interesting representation.**
- ★ **Save and close your new world when complete.**

AgentSheets Tutorial2: Reuse from Ozone (Example Specialized Model)

Reusing Ozone World to Create Photosynthesis World Simulation

The new environment that you could simulate is an instance of Photosynthesis. This simulation will involve the interaction of several complex factors. There will be a sun, the sun produces solar energy energy as sunrays, the sunrays shine on plants, and after they hit the plants, the plants grow bigger because the sun's energy causes photosynthesis to occur.

- On the next sheet **draw** a simple picture of what you would expect a Photosynthesis Simulation to look like. Also identify candidate **agents** for your simulation.

(Take 2-5 minutes brainstorming and drawing.)



Creating new agents and Adding behaviors to create Photosynthesis Simulation.

A simulation in Agentsheets is simply a set of agents that work together to create visual effects.

A key aspect of reusing a simulations is to specialize existing agents and adding new agents. We will begin with the small task of reusing a new agent from the Ozone World.

Task 1. Click on Gallery

- Click on **File** → **Save**.
 - Name this gallery as **Photosynthesis_Gallery_YourInitials**
- Create a new Worksheet**
- Click on **File** → **New Worksheet**.
 - Name this worksheet as **Photosynthesis_Worksheet_YourInitials**

Task 2. Draw a new agent

 Refer to interactions guide for **Gallery/Worksheet Tools** and more help.

Task 3. Reusing and creating new behavior for your new agent

- Some agents already have behaviors. You may need to look at there behaviors to get started. You can reuse their behavior by just dragging the entire rule that you would like to reuse into your agent's behavior palette.
- Add new behaviors.

 Refer to interactions guide for more **Help**.

- **Think of other interactions to make your *Photosynthesis World* simulation an interesting representation.**

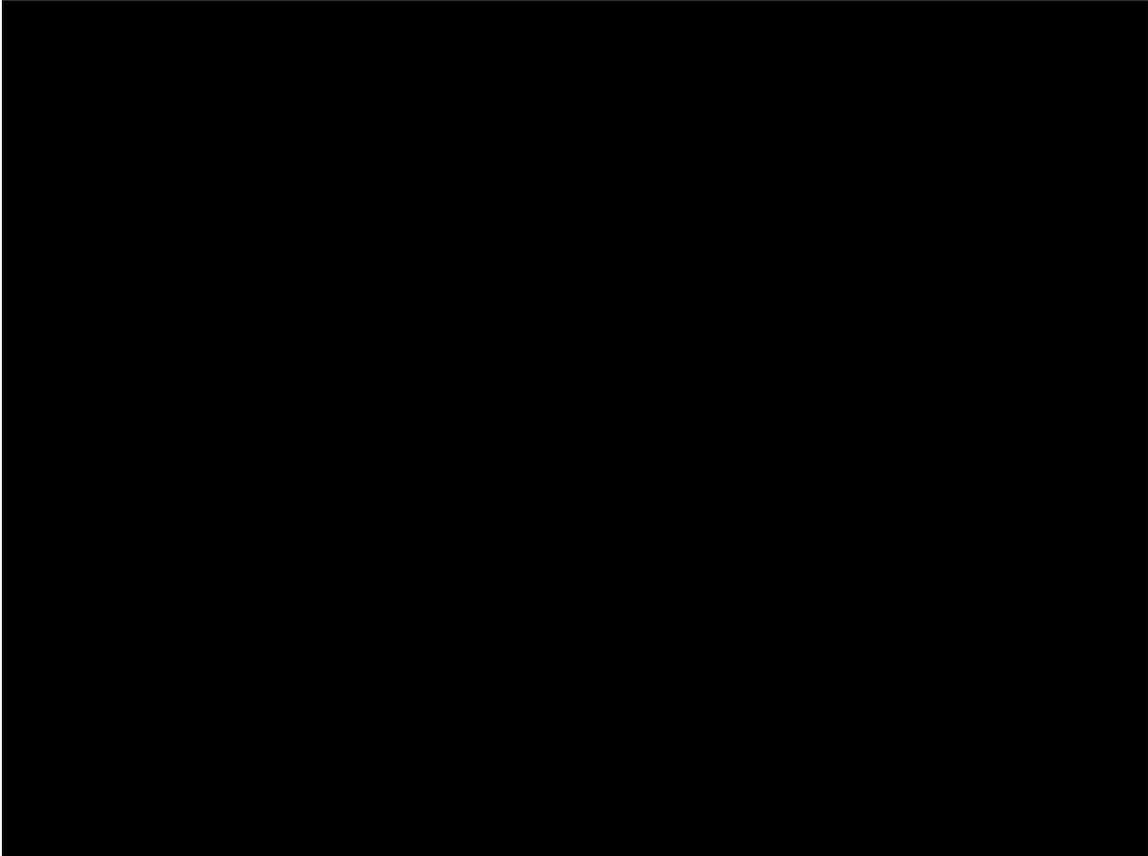
- ★ **Save and close your new world when complete.**

Reusing Starter World to Create Photosynthesis World Simulation

The new environment that you could simulate is an instance of Photosynthesis. This simulation will involve the interaction of several complex factors. There will be a sun, the sun produces solar energy energy as sunrays, the sunrays shine on plants, and after they hit the plants, the plants grow bigger because the sun's energy causes photosynthesis to occur.

- On the next sheet **draw** a simple picture of what you would expect a Photosynthesis Simulation to look like. Also identify candidate **agents** for your simulation.

(Take 2-5 minutes brainstorming and drawing.)



AgentSheets Tutorial2: Interaction Guide



Interaction Guide (Creating Agents)



To **inspect** an agent more closely **double click** on it and you will see its 32x32-pixel representation. If you don't get a larger view of the agent that is editable, you are in the wrong view.

- Select **Gallery** → **Designer Views** to get the correct view.



Drawing an agent

- Click on **Gallery** → **New Agent** and add a **agent** by entering it in the dialog box.
- Double click on **agent** at the bottom of the **gallery** to select it. Use the pencil and color palette to draw your new agent.



After creating a new depiction you always want to define a **Mask Color** for it. Masks are like filters or films of a given color that cover the cell a depiction inhabits. The reason you may need this is the following example:

When a **cloud** floats through the **sky** it may float in front of the **sun**. Instead of blocking out the **sun** you can still see portions of the sun this is caused by having a transparent background. If the background were not transparent the **sun** would be obscured from view completely when a **cloud** passes over it.

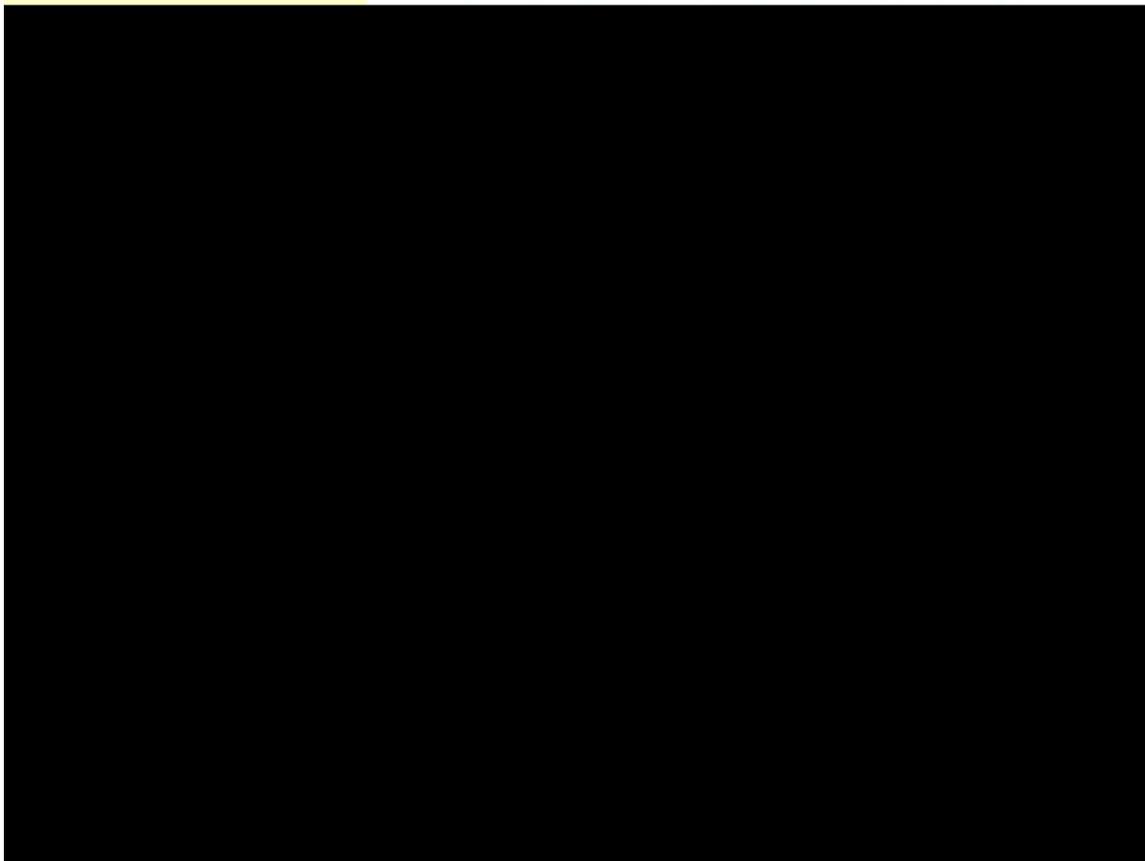
For a transparent background mask

- Click on **Gallery** → **Mask Color** and then select **upper left pixel** (is the most common mask).



To test one rule at a time you can grab the whole rule and drag it on top agent in the worksheet. If you do this correctly the area around the agent will be highlighted with a red box.

You can use  **Run** to test the simulation, but to incrementally run the simulation one step at a time you should use the step button  instead.





Interaction Guide Behavior Interactions



(Condition Commands)



The **See** condition command looks at any of its eight immediate neighbors (at a depth of 1 cell away) and perhaps itself to see if the indicated cell contains the depiction shown in the depiction window of the command.



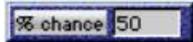
The **See a** condition command looks at any of its eight immediate neighbors and itself to see if the cell indicated by the direction operator contains an agent of the class specified in the class menu of the command. This command picks out all the depictions of a given agent class. For example, in the Gallery below, all 4 depictions shown are members of the class Auto. The name of an agent class is the same as the name of the base agent depiction of that class.



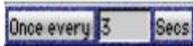
The **Next to** condition command contains a comparator menu, a number field, and a depiction menu, in that order. These fields combine with the title of the command to form a coherent phrase: "next to greater than 3 north-facing auto(s)."



The **Empty** condition command checks to see if its immediate neighbor or itself (as specified by the direction operator) is empty.



The **% Chance** condition command as shown above returns a value of T or True 50% of the time. The % Chance command chooses a random number between 1 and 100 inclusive whenever it is executed. If this number is less than the number indicated in its number field the command returns a value of T or True, indicating the condition was met. If the random number chosen is greater than or equal to the number shown in the command's number field, the command returns a value of NIL or False, indicating the condition was not met. You type in the %chance of the command returning true in the command's number field.



The **Once every** (number) **Secs** command as shown to the left checks to see if 3 seconds has passed since it was last checked.

(Action Commands)



The **Move** action command tells the executing agent to move one cell in the direction indicated by the direction operator.



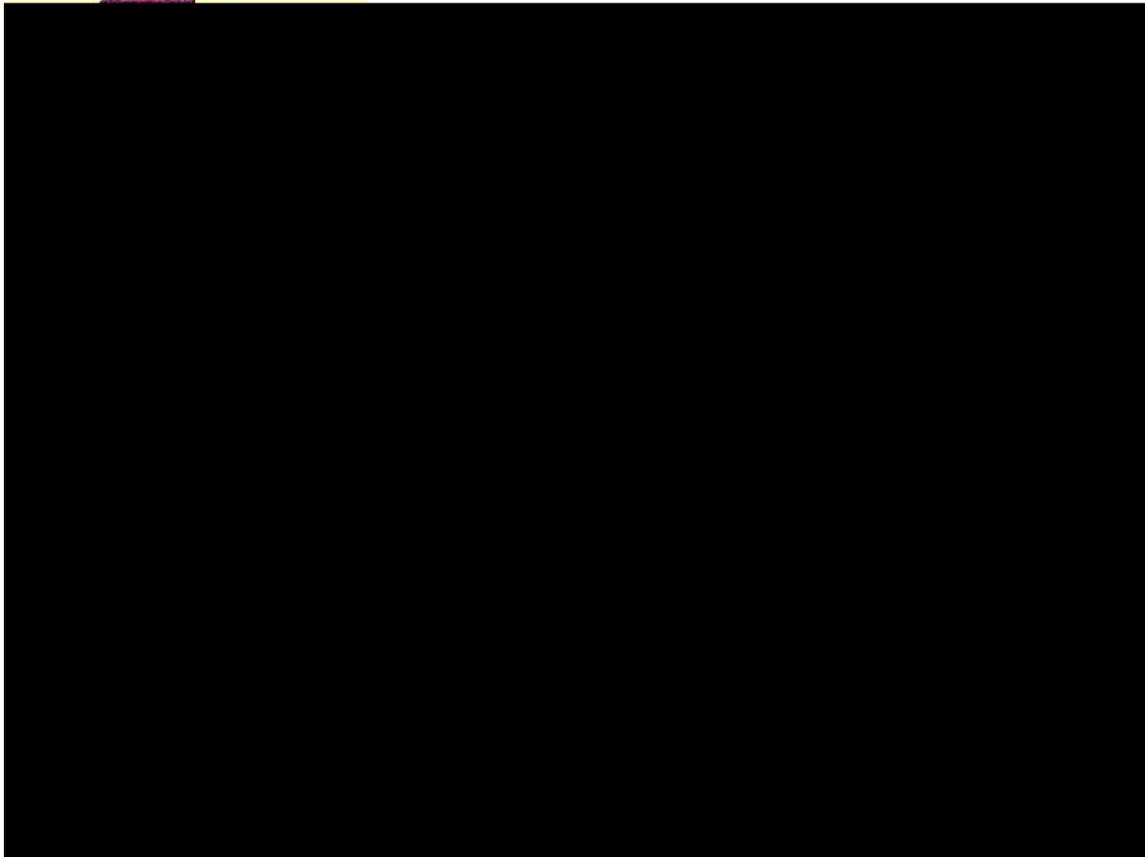
The **Change** action command changes the depiction of the agent in the cell indicated by the direction operator to the depiction displayed in the command's depiction window.



The **New** action command makes a new agent in the cell indicated by the direction operator.



The **Erase** action command erases the agent located in the cell indicated by the command's direction operator.



Appendix H: SimBuilder Learning and Reuse Tutorials
SimBuilder Tutorial Learning Session

Exploring Visual Programming

Squeak SimBuilder Tutorial
Implementing a Model of the Water Cycle

Human Computer Interaction
@
Virginia Polytechnic Institute and State
University

This tutorial is a draft of materials being developed as part of behavioral research underway in the Computer Science Department at Virginia Tech. It is provided on an "as-is" basis; however, we welcome comments and suggestions. Please direct any feedback to seals@csgrad.cs.vt.edu.

SimBuilder Tutorial © Virginia Tech Visual Languages Group

Exploring the Water Cycle Model

- **Double click** to open the Squeak environment.



- **Select Water Cycle**



- This will open your first example of a simulation.
- Now, **Press**  **to start** this model.
 - Watch the Simulation.
 - What actions are taking place?
- Press  after a few minutes of observing the model.

Changing The Behavior of Clouds

- Examine some of the more complicated behaviors by selecting the *Cloud* and reviewing its behavior.

 *Take a moment to review the interaction guide to gain a better understanding of the interactions between player's behaviors*

Make Clouds more active.

Currently the behavior of the cloud is to move forward 5 spaces. Find the **script** that causes this behavior. Let's make the clouds move a lot more.

- Select the desired script  **cloud moving** and drag it to a clear workspace.
- Increase the value for *moving forward* to **10**.
 - Press  to see how your changes affect the simulation.
 - Press  and try another change.

Make the clouds move vertically.

Currently the behavior of the cloud is to move horizontally across the sky. Let's try to change the behavior of the cloud so that it will move vertically.

- <Alt> Click** or **<Middle button> Click** your cloud and its *Halo* will appear.
- Select **Rotate** and move your cloud just a tiny bit for its direction arrow to appear.

In order to change the direction that the *player* moves

Click on the **green arrow** and change its direction so that it points to the **down**.

- Press  to see how the simulation has changed.



Giving the Bird Behaviors

In this playground, we want the bird to be able to fly through the sky.

- Double click on your **bird** and its **viewer** will appear.
 - ◆ The **viewer** is a window to select behaviors for your player.
 - ◆ Let's add the behavior that your player will move in one direction.
 - ◆ Select



- ◆ and Drag it out of the **viewer** and place it anywhere, preferably not in the playground.
- **Press**  to see how your bird acts within the playground. Your bird should fly across the playground. If it doesn't you may need to check out which direction your bird is flying.
- **<Alt> Click** or **<Middle button> Click** your bird and its **Halo** will appear.



- **Select Rotate** and move your bird just a tiny bit for its direction arrow to appear.

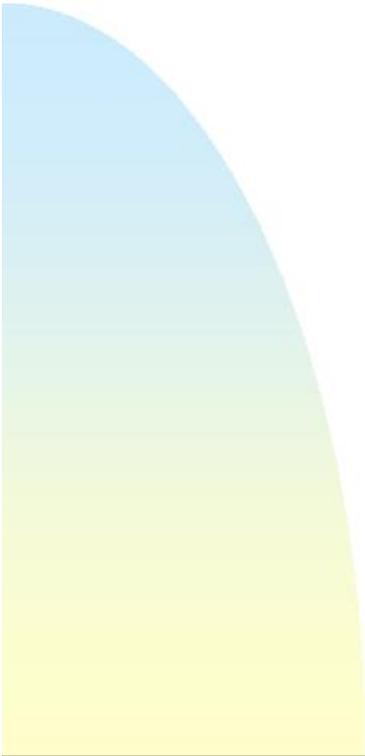
- Click on the **green arrow** and change its direction so that it points to the right.



Try putting a bird on the ground. Does it move? What would you need to make it move?

Now you have all the basic tools you need to create your own Simulations!!

Volcano Simulation



Squeak SimBuilder Tutorial © Virginia Tech Visual Languages Group-Draft



Adding Behaviors to Volcano Simulation

- To make your volcano erupt, **players** need to interact with each other. Add actions and behaviors to your simulation in your next session, we would like you to think about the possible behaviors that your **players** can possess.

For example, in the simulation that you reviewed, a **cloud** moves from place to place, produces rain and changes itself to a **rain cloud**.

- Think of interactions that happen to cause a volcano to erupt. The eruption of a volcano is caused by pressure within the earth crust that needs to be released.
- **Have fun trying to get your players to collaborate in interesting ways.**

When finished Save your project.

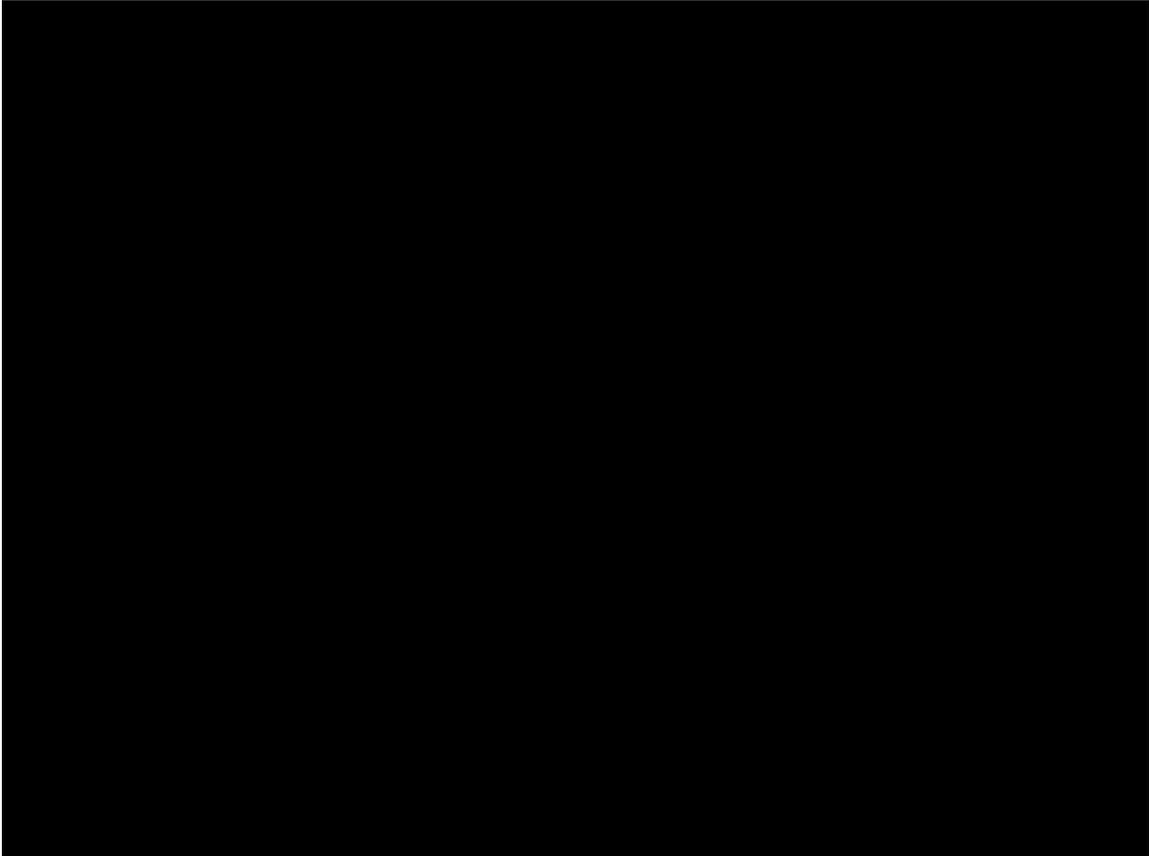
Press **PUBLISH IT!** in the Navigator and then **Saves on local disk only**

Press **< PREV** project and you should be back at the Welcome page.

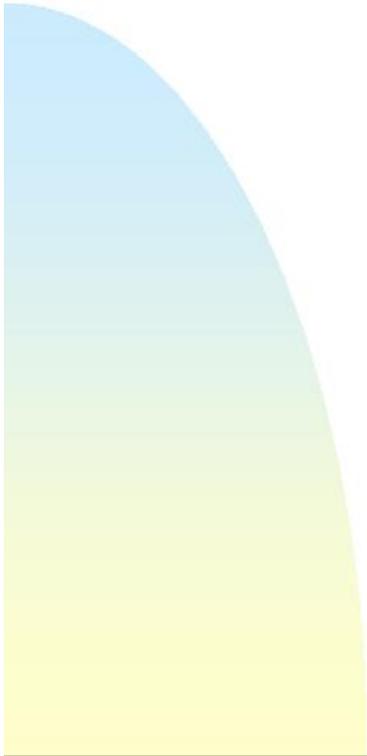
SimBuilder Tutorial2: Reuse from Starter (Component Generic Model)

Exploring Visual Programming

Section II



Ocean World Simulation



Creating new players and Adding behaviors to create Ocean World Simulation.

A simulation in SimBuilder is simply a set of players that work together to create visual effects.

A key aspect of reusing a simulation is to reuse existing players and add new players. We will begin with the small task of reusing a new *player* from the Starter World.

Task 1. Create a new project

In the **Navigator** and Press **NEW** project.

- Click **Unnamed1** at the bottom of the new window and
- Rename it **OceanWorksheetYourInitials**

Click the **Ocean** Project to enter it.

Task 2. Reusing and creating new behavior for your new player

- Some player already have behavior scripts. You may need to look at their behaviors to get started. You can also reuse a player by just using the grey *repaint brush* to change their look and just reuse its behavior



Task 3. new players



Refer to interactions guide if you need *Help*.

- Add behaviors for new players you create.  Refer to interactions guide for *Help*.

Think of other interactions to make your Ocean World simulation interesting.

- Press **PUBLISH IT!** and then **Save on local disk only**
- Press **< PREV** and return to the Welcome page.

SimBuilder Tutorial2: Reuse from Ozone (Example Specialized Model)

Reusing Ozone World to Create Photosynthesis World Simulation

The new environment that you could simulate is an instance of Photosynthesis. This simulation will involve the interaction of several complex factors. There will be a sun, the sun produces solar energy energy as sunrays, the sunrays shine on plants, and after they hit the plants, the plants grow bigger because the sun's energy causes photosynthesis to occur.

- On the next sheet **draw** a simple picture of what you would expect a Photosynthesis Simulation to look like. Also identify candidate **players** for your simulation.

(Take 2-5 minutes brainstorming and drawing.)



Reusing Ozone World

In the Ozone depletion Cycle a factory emits CFC into the atmosphere and a heterogeneous reaction takes place. This reaction converts the inactive chlorine and bromine reservoirs to more active form. No ozone loss occurs until sunlight initiates the catalytic ozone destruction.

- Open the Ozone Depletion Simulation.
- Now, **Press**  **to start** this model.
 - Investigate each player to discover its' behavior.



The Smoke_stack emits chemicals into the atmosphere.



The Chemicals are emitted by the smoke stack and move up into the atmosphere. They are changed into active BrCl when contacted by the sun.



The Sun replaces the inactive chemicals with active BrCl.



The BrCl moves randomly until it contacts an ozone player.



The ozone absorbs (erases) BrCl and is changed into a weaker ozone

- Press  after a few minutes of observing the model.



Refer to interactions guide for *Help*.

SimBuilder Tutorial2: Reuse from Ozone (Example Specialized Model)

Reusing Ozone World to Create Erosion and Ocean World Simulation

The new environment that you could simulate is an Ocean biosphere. This simulation will involve the interaction of several complex factors. There will be an ocean, the ocean produces waves, the waves hit the beach, and after they hit the beach they cause the amount of the sand on the beach to decrease from erosion...

- On the next sheet **draw** a simple picture of what you would expect an Ocean World Simulation to look like. Also identify candidate *players* for your Ocean World simulation.

(Take 2-5 minutes brainstorming and drawing.)

Creating new players and Adding behaviors to create Ocean World Simulation.

A simulation in SimBuilder is simply a set of players that work together to create visual effects.

A key aspect of reusing a simulation is to reuse existing players and add new players. We will begin with the small task of reusing a new *player* from the Ozone World.

Task 1. Create a new project

In the **Navigator** and Press **NEW** project.

- Click **Unnamed1** at the bottom of the new window and
- Rename it **OceanWorksheetYourInitials**

Click the **Ocean** Project to enter it.

Task 2. Reusing and creating new behavior for your new player

- Some player already have behavior scripts. You may need to look at their behaviors to get started. You can also reuse a player by just using the grey *repaint brush* to change their look and just reuse its behavior



Task 3. new players

 Refer to interactions guide if you need *Help*.

- Add behaviors for new players you create.  Refer to interactions guide for *Help*.

Think of other interactions to make your Ocean World simulation interesting.

- Press **PUBLISH IT!** and then **Save on local disk only**
- Press **< PREV** and return to the Welcome page.

SimBuilder Tutorial2: Reuse from Starter (Component Generic Model)

Reusing Starter World to Create Photosynthesis World Simulation

The new environment that you could simulate is an instance of Photosynthesis. This simulation will involve the interaction of several complex factors. There will be a sun, the sun produces solar energy energy as sunrays, the sunrays shine on plants, and after they hit the plants, the plants grow bigger because the sun's energy causes photosynthesis to occur.

- On the next sheet **draw** a simple picture of what you would expect a Photosynthesis Simulation to look like. Also identify candidate **players** for your simulation.

(Take 2-5 minutes brainstorming and drawing.)

Creating new players and Adding behaviors to create Photosynthesis Simulation.

A simulation in SimBuilder is simply a set of players that work together to create visual effects.

A key aspect of reusing a simulation is to reuse existing players and add new players. We will begin with the small task of reusing a new **player** from the Starter World.

Task 1. Create a new project

In the **Navigator** and Press **NEW** project.

- Click **Unnamed1** at the bottom of the new window and
- Rename it **PhotosynthesisYourInitials**

Click the **Photosynthesis** Project to enter it.

Task 2. Reusing and creating new behavior for your new player

- Some players already have behavior scripts. You may need to look at their behaviors to get started. You can also reuse a player by just using the grey **repaint brush** to change their look and just reuse its behavior



Task 3. new players.

 Refer to interactions guide if you need **Help**.

- Add behaviors for new players you create.  Refer to interactions guide for **Help**.

Think of other interactions to make your **Photosynthesis** simulation interesting.

- Press **PUBLISH IT!** and then **Save on local disk only**
- Press **< PREV** and return to the Welcome page.

SimBuilder Tutorial: Interaction Guide



Interaction Guide (Object Halo & Handles & Paint Tools)



Halo Tools



To manipulate objects in Squeak *SimBuilder* select the object and **<Alt> Click** or **<Middle button> Click** your object and its *Halo* will appear.

-  The **pink Close handle** will move your object to the trash.
-  The **red Menu handle** will open a menu of other options for your object.
-  The **black Pick Up handle** will let you *Lift Your Object* and move it.
-  The **brown Move handle** will let you *Drag* your object.
-  The **green Duplicate handle** will let you *Copy* your object.
-  The **light grey Debug handle** is used for script debugging.
-  The **grey Repaint handle** will let you *Repaint* your object.
-  The **purple Change Color handle** lets you *change* the color of your object.
-  The **yellow Change Scale handle** will let you *Resize* your object to make it larger and smaller.
-  The **dark yellow Make a tile representing this object handle** will make a *Label* for this object.
-  The **light blue Open a viewer of Me handle** will let you view the characteristics of an object.

Paint Tools

Just click on the Paint brush and Paint tools will appear.



- Use the **Paint Brush** to create.
- Use **Paint Bucket** to fill areas.
- Use **Dropper** to select a color.
- Use **Eraser** to modify.
- **Multiple Circles** choose brush size.
- **Color palette will change color.**
- Press **Keep** when complete.

Appendix I: Post-Questionnaire

Spring 2003 Simulation Questions

SimID

Please respond by circling the reaction that best reflects your reaction to the system:

Terrible ----- Wonderful

1 2 3 4 5

Frustrating ----- Satisfying

1 2 3 4 5

Dull ----- Stimulating

1 2 3 4 5

Difficult ----- Easy

1 2 3 4 5

Rigid ----- Flexible

1 2 3 4 5

Boring ----- Fun

1 2 3 4 5

Please respond by selecting the reaction that best reflects your impressions:

1. This system was easy for me to learn and use.

Strongly Agree Agree Neutral Disagree Strongly Disagree

2. It was easy to get started.

Strongly Agree Agree Neutral Disagree Strongly Disagree

3. It was difficult to remember where some of the tools and commands were located.

Strongly Agree Agree Neutral Disagree Strongly Disagree

4. This system would be easy to use by folks who don't know much about computers.

Strongly Agree Agree Neutral Disagree Strongly Disagree

5. This system would be fun for building simulations.

Strongly Agree Agree Neutral Disagree Strongly Disagree

6. I have a good understanding of how to use this system to build simulations.

Strongly Agree Agree Neutral Disagree Strongly Disagree

7. I was able to use this system to turn my ideas into working simulations.

Strongly Agree Agree Neutral Disagree Strongly Disagree

8. Turning my ideas into working simulations was complicated.

Strongly Agree Agree Neutral Disagree Strongly Disagree

9. To create a new object (e.g. bird). How many steps would this take?

1 2 3 4 5 or more

How many steps would be appropriate to keep your interest and lower the risk of frustration.

More Steps No Change Less Steps

10. To give the bird behaviors, how many steps would this take?

1 2 3 4 5 or more

How many steps would be appropriate to keep your interest and lower the risk of frustration?

More Steps No Change Less Steps

11. Creating visual rules by dragging and dropping the desired parts to create behavior was complicated.

Strongly Agree Agree Neutral Disagree Strongly Disagree

12. Rule ordering was confusing.

Strongly Agree Agree Neutral Disagree Strongly Disagree

13. My simulation works logically, but the tools made it hard to create the desired behavior

Strongly Agree Agree Neutral Disagree Strongly Disagree

14. It was hard to recover from errors.

Strongly Agree Agree Neutral Disagree Strongly Disagree

15. I was able to use this system to turn my ideas into working simulations.

Strongly Agree Agree Neutral Disagree Strongly Disagree

16. I found the creation of rules for object behaviors confusing.

Strongly Agree Agree Neutral Disagree Strongly Disagree

17. The rules I created for objects' behaviors were simple and natural.

Strongly Agree Agree Neutral Disagree Strongly Disagree

18. Reusing parts of the Ozone World to build a new world was straightforward.

Strongly Agree Agree Neutral Disagree Strongly Disagree

19. I understood the behavior of the objects in Ozone World with little investigation.

Strongly Agree Agree Neutral Disagree Strongly Disagree

20. Reusing parts of the Starter World to build a new world was straightforward.

Strongly Agree Agree Neutral Disagree Strongly Disagree

21. I understood the behavior of the objects in Starter World with little investigation.

Strongly Agree Agree Neutral Disagree Strongly Disagree

22. I was able to have agents any size I wanted.

Strongly Agree Agree Neutral Disagree Strongly Disagree

23. At this point, I am enthusiastic about creating new simulations.

Strongly Agree Agree Neutral Disagree Strongly Disagree

Explain the behavior of each of the following objects. (Note: see provided graphic). SmokeStack, Chemicals, Ozone, Sun

Explain the behavior of each of the following objects. (Note: see provided graphic). Emitter, Mover, Changer, Eraser

The visual representation or picture is a good match with the behavior of the object.

Strongly Agree Agree Neutral Disagree Strongly Disagree

Please answer the questions below.

What was most interesting or fun?

What was least interesting or fun?

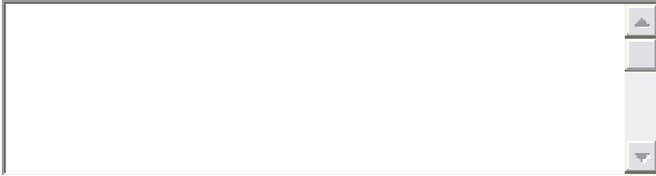
Did you find the example simulations used in the tutorial effective? Why or why not?

Did you find the instructions in the tutorial helpful? Why or why not?

What 1-2 things would you change if you were asked to revise the tutorial?

Suppose you were going to build a computer simulation of a volcano exploding for earth science. What sorts of things do you think would be involved (i.e. what objects and what do they do)?

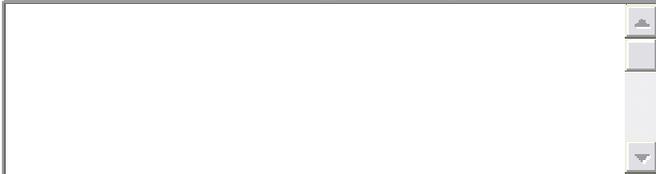
As well as you can, please describe what you think is the best way to come up with projects? (What criteria would you emphasize?)



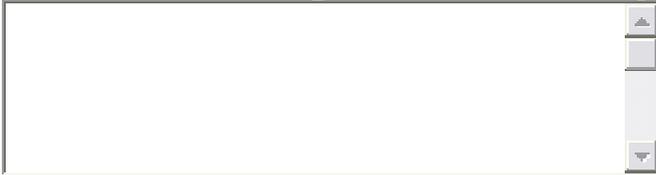
How could you use this type of software?



Were you able to reuse the rules of any other agent? If so did this aid you in the creation of new agents in your simulation? Yes or No? Please explain.



Which operation is easier? Creating new agents and functionality from scratch or to reuse agents and functionality. Please explain.



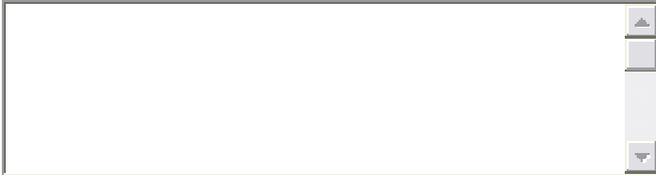
Can you think of any steps that would have made the reuse activity more straightforward?



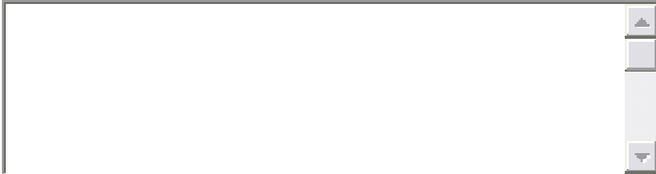
Would you be willing to use such a tool for creating educational simulations for use in your classroom? Please explain.



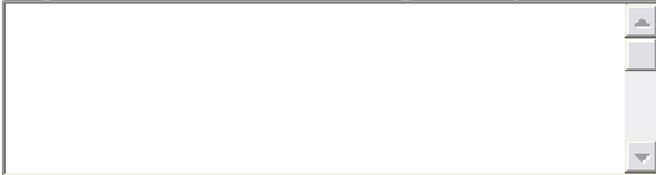
What ideas do you have for simulations that would be useful for you or someone in your discipline?



Can you think of any changes or enhancements to this system, especially ones that would make it more useful in creating simulations for novices? Please briefly describe the features that you think are needed in building simulation software.



Any final comments about your experiment activities or the software.



Appendix J: Post-Questionnaire Data and Statistical Analysis

The next couple of pages contain user comments about the environments from the Post-Questionnaire.

What was most interesting or fun?

Comments about AgentSheets

- Drawing & coloring items
- Creating agents for the simulation, Creating the scenario and watching it play out
- The more I learned the program (how to do the representation and make them move the way I wanted them to) the better I felt about my skills and abilities.
- Actually creating the simulation, having a real life problem/event and visually seeing the process
- It was fun working on the simulation and seeing it put to action when the process was over.
- Working on the photosynthesis model
- I really liked seeing what I did actually work. Also, it was fun to see what I could see in my mind on there screen.
- I think that this program is interesting and it was neat to see ideas into motion.
- Getting to see the final results

Comments about SimBuilder

- It was interesting and fun to create and object and place rules with it to create a simulation.
- The Volcano simulation where the "sparks" moved from the volcano's plume.
- Seeing the objects.
- Creating the volcano
- Reusing the emitter and things to create a beach.
- Yes, but a little frustrating at times...

What was least interesting or fun?

Comments about AgentSheets

- Creating the behaviors.
- Trying to get the 2 separate agents to interact
- Becoming familiar with techniques & procedures
- Adding behaviors so that they actually did what I wanted
- Trying to figure out how the different keys worked
- Figuring out the program and how to go about drawing.
- Trying to create rules/behaviors for the objects, the condition and attribute menu were hard to use,
the menus were based on very specific situations, not very user friendly at first but after more exposure
the program was easier to use especially with help of instructor
- The initial intimidation I have when I work with computers.
- Getting started and learning the way around

The drawing part. It was hard to actually draw what I wanted. The drawing part was a little rigid. For me it was frustrating because I tend to learn better with a combination of auditory and visual for the first time with a program. Also you could not change the picture sizes.

Adjust the behavior

Comments about SimBuilder

- The least interesting was figuring out how to correct an error.
- The simulation involving the birds moving
- Operating with the rules.
- Changing the ocean erosion
- The system freezing
- trying to figure out how to apply the rules/behaviors exactly as i wanted

Did you find the instructions in the tutorial helpful? Why or why not?

Comments about AgentSheets

Yes, it cleared up some confusion, especially the graphical representations of the tools in the various boxes (e.g., behavior, new agent, edit depiction, etc.).

Yes-straightforward

Instructions sometimes difficult to understand & follow. This could present challenges to people who do not have strong computer skills.

Yes, but the behaviors were complex enough that it took trial and error.

Sort of, but personal help was better

I found them to be helpful. The screen captures could have been more clear on the instructions.

But I understand that is how it is sometimes.

Yes, allowed you to become familiar with the program, how to use it, and the different features of the program, tutorial was necessary in order to create your own simulation

Yes, the directions did a good job walking me through the exercise, I am not good at discovering computer steps on my own.

They were a little confusing but I thought the tools were also

for the most part, then did help. Sometimes, the instructions assumed I knew exactly where to go.

So more details might be a good thing to have. For example, describing everything word for word

Yes but it helped when the instructor also walked through some of the things as well.

They appeared to need more detail, they did however give me a better understanding in the specific areas

Comments about SimBuilder

Yes, however, it was difficult to figure out how to correct an error.

The instructions seemed to confusing

Many details were missing from the tutorial.

Yes - I was able to navigate effectively

Yes, very straightforward

Yes, but I still had a little trouble figuring out how to apply the instructions

What 1-2 things would you change if you were asked to revise the tutorial?

Comments about AgentSheets

Show it more in a step-by-step layout.

Size of graphics, graphics no so blocky looking make it so I can create one picture & have everything

interact

Make instructions easier to understand for those not computer literate, include more graphics, use larger

fonts for ease in reading

1. Make the size of agents changeable after they're created 2. simplify the behaviors.

Just refine the beginning directions

I think that there were too many activities to do. By the third one I was ready to go because they were so

complicated. I was used to seeing things in real life view versus what I drew so at times it was hard to put

things together.

Change the menus to add behaviors to the object, make them more self explanatory and easier to use,

less specific with directionality

I really don't have any suggestions on how to change the tutorial.

Have the desktop close when it opens make the tools better

Does it only have to be science? Maybe you can use another content as an example. It would be

interesting to see how English is applied.

If there could be a way to add sound to the tutorial that would be helpful.

I would make the behavior boxes clearer, I felt they were very hard to use. The dragging of boxes confused me as well. However, by the end I felt more comfortable with using the program.

Comments about SimBuilder

I would place more instruction about the rules and how to use them.

Simplify the steps.

Make the tutorial more detailed-with step by step screen shots. Make the rules for getting things to more less complicated.

The ocean erosion was a bit confusing

I don't know that I'd change anything, but I did appreciate the helpful guidance from the facilitator. This helped when I was lost or confused

Suppose you were going to build a computer simulation of a volcano exploding for earth science. What sorts of things do you think would be involved (i.e. what objects and what do they do)?

Comments about AgentSheets

Geology, population density (e.g., human, animal, etc.), forestry, oceanography, etc. --- stages of volcanic eruption, lava, scenery, etc.

magma -> shoot out, lava- flow down, have things melt & bubble

sketch simulation, what do volcano, steam, lava, heat, sparks, etc., do (roles)

A non-moving mountain flowing, lava, rising magma, rising smoke, and flashing lightning.

Mountain, Lava, rocks, ...explosion

A volcano, lava, smoke, dark clouds in the sky. The lava would come out of the volcano, the sky would have the dark clouds around it, and smoke would come out of the volcano as well.

Pressure would build up under volcano, volcano would erupt lava, lava would produce heat, lava would produce smoke

Volcano, atmosphere, sun, ground, sky, lava, changes in the environment around the volcano like the color of the land.

Mountain lava smoke sky

Volcano, ash, lava, ground, and gray background. The volcano shaking and the lava coming out of it would be cool too.
Need a background, then a mountain. the lava would be coming out of the volcano because of pressure.
the lava would shoot up the volcano, erupt down the sides of the volcano and onto the ground. there would be sparks that also are emitted from the top of the volcano.
I would include lava or sparks hitting the volcano and show the sky becoming darker .

Comments about SimBuilder

I would have to use the paintbrush to create a volcano, sparks, and background scenery. The sparks would fly out of the volcano to simulate explosion.
Volcano, lava, sparks of lava rock, black and greyish colored smoke
The things that were in the tutorial were fine. I cannot think of any more.
Mountain - explosion of rock and lava an ocean that surrounds the mountain upward arrows to stimulate building pressure
Volcano, earth, smoke, fire, sparks, lava
I'd build volcano, lava, and lava spark objects. the lava spark objects would rise out of the volcano

As well as you can, please describe what you think is the best way to come up with projects?
(What criteria would you emphasize?)

Comments about AgentSheets

The best way to "come up with projects" is to tap into student's passions, to emphasize their natural talents and gifts.
Math games (simulations of distance traveled, estimation, etc to find projects, look at some lessons on the web or in teacher manuals
think through, visualize (sketch on paper), experiment, revise as developing
Brainstorming, looking @ current periodicals. I would emphasize relevance to content matter. periodicals.
Through imagination, and interest
I would look at the most complicated areas of study and then create projects.
Tying them to curriculum (ex. what is being studied in science) use the simulations to aid visual and kinesthetic learners and to help explain difficult science processes
Emphasize systems that are talked about in the classroom, make sure that the project incorporates multiple steps and agents.
Imagination
See what interests students, and good research. Then create a lesson/project that will appeal to the students. Something they can relate to.
I think that the best way is to see what subjects are going to be taught and what information is going to be given out. From there brainstorm to see if there are projects that could help the students

learn
if they could see a visual of it.

Comments about SimBuilder

I would emphasize the use of rules to create movement.

The best way to come up with projects should involve instructions that will take no more than 10-15 minutes to figure out since class time will be usually around 50-60 minutes. Also, students tend to tune out if instructions are too long and tedious.

Having people get in a group to brainstorm.

Using real life occurrences. historical events such as volcanoes, earth quakes etc.

take the course outline and curriculum and come up with examples for lectures

brainstorming activities that relate to real-world experiences

How could you use this type of software?

Comments about AgentSheets

In social studies it would be near impossible, for example, how would I represent the Columbian exchange (e.g., plant and animal migration) from 1492 to 2002?

Reinforce concepts taught in class: to promote inquiry based learning

Unsure at present time

Perhaps. If it's limitations would not limit the expression or understanding of concepts.

To help students create their own simulations for better understanding of the subject matter

I could use this type of software for cause and effect relationship representations.

Mainly for science processes

I could use it to describe an action like the growth of a city or the pollution of the environment for the

class before a lesson, or I could have a class do research and come up with their own simulation as a

final project.

For simple simulations

I don't know how I could use this software for English. Maybe I could use it in describing a scene in

a novel or play, or I can use it to teach grammar lessons. Actually, now that I think about it, this software

can help me give students more visuals.

To show students how something works. Sometimes it is very helpful for people to see something.

I would use this to help students investigate weather conditions or cause and effect problems

Comments about SimBuilder

I would use this type of software to show my students how things occur in nature or how something functions. I would use this software as a form of interactional and visual learning.

This software is useful in an earth (geology) science class.

This is good for science class and maybe math class in order to make the "what happens next" step be visual.

I would use it in a small classroom setting. I would also use it as a tutorial for other applications such as breast self exams etc

in a science class

this SW would be good for teaching earth and biological sciences in grade school

Were you able to reuse the rules of any other agent? If so did this aid you in the creation of new agents in your simulation? Yes or No? Please explain.

Comments about AgentSheets

No, because I wasn't sure how to get to the other agents. Whenever I went to "File", I was stopped from

accessing "Open". Plus, I enjoyed creating the agents.

Yes -> by the 3rd agent I felt pretty comfortable

Yes, It was helpful to refer back to when developing new agents behaviors

Yes it was the most useful thing I learned how to do. It minimized trial and error with behaviors.

Yes, I could reuse drawn objects

I was able to think of how the rules from one would help me, but I did not reuse rules exactly. I always

had to change them to fit what I was doing.
 no, i couldn't figure out how to incorporate them into the ideas that I had formed
 Yes,, i reused the rules of the last simulation to help me with the photosynthesis simulation. I reused
 the emitter, and the replacer and the random acts.
 no the eraser didn't work properly
 yes I was able to reuse rules, and it was nice to have because it was already done for me. Also,
 now
 I know how to make waves move.
 Yes, Yes It was helpful to reuse things already done. If I couldn't use it exactly as it was set up I was
 able to see how it was already set up and then tailor to my need.
 yes it saves time in having to redraw or direct objects

Comments about SimBuilder

Yes, I was able to reuse rules and this helped make the creation of my simulation faster and less confusing.
 Yes. The emitter function proved useful. It saved time in trying to program an object to move.
 Yes, this was an aid because I did not have to make up the rules for that object.
 yes
 yes. i reused alot in the emitter simulation. that was straightforward to follow
 yes. yes. i could simply copy rules instead of having to figure out how to write new ones. this was a bit confusing, but

Which operation is easier? Creating new agents and functionality from scratch or to reuse agents and functionality. Please explain.

Comments about AgentSheets

Creating from scratch.
 Creating new -> you don't have to mess w/someone else's work
 Modify behavior? Much easier to reuse agents & modify behavior?
 Reusing, for the reason listed above. Drawing is fun,, behaviors were a little tedious.
 Creating new ones is more fun, I like it better
 Creating new agent from scratch. This is because by looking at what the example it I would get confused at to what I was asked to do.
 Creating new agents was easier because they were self drawn based on your own ideas
 They both seemed to be easy once I learned how to work the simulation.
 reusing agents was better
 Creating my agents, but reusing the rules
 Reuse- I had a better concept of what something was
 both became equally easy as time progressed, at first I would say creating a new agent was easier

Comments about SimBuilder

I thought it was easier to reuse the agents because the rules were already set in place.
 Resuing agents and functionality is easier because it can save you a few steps and give you more time for creativity in your construction of a simulation.
 To reuse agents was better. It saved time as well as frustration.
 creating new agents - because i do not have to figure out what the other person previously constructed and why REUSE! the volcano example wasn't as starightforward b/c i didn't really understand how to create rules. but in the

emitter simulation i was able to reuse existing rules. it saves alot of time.

reusing agents is definitely easier. i didn't want to have to think about how to develop new ones. this seemed a bit complicated. (perhaps if i had better experience w/ the SW, i'd be more confident.)

Can you think of any steps that would have made the reuse activity more straightforward?

Comments about AgentSheets

No

Make it easier to reopen previous galleries to access agents

No. It was quite easy.

Make sure the ask activity I would do and the example are very similar so I could reuse rules (behaviors and actions).

I think that the thing that made the reuse activity straightforward was when I used it and practiced it.

Not really

No, it is pretty straightforward

Not at this time

Maybe have a reuse link as an option in the gallery

Comments about SimBuilder

N/A

No

Having that stated in the tutorial to look at the objects as you can use them by copying over them and retitling them to do what you want for the project you need.

An explanation of the construction

Better names

I don't know...

Would you be willing to use such a tool for creating educational simulations for use in your classroom? Please explain.

Comments about AgentSheets

In social studies it would be near impossible, for example, how would I represent the Columbian exchange (e.g., plant and animal migration) from 1492 to 2002?

Yes I think simulations are helpful visual tools for when you are explaining something and cannot leave the school

Yes when appropriate

Yes, but only if I could change the things listed above (1. Make the size of agents changeable after

they're created 2. simplify the behaviors.)

Yes, especially for upper grades

I would be willing if I had extensive training of how to create things that are eye catching and interest gaining.

If I could not get the proper training I do not think that I would use this software personally.

Yes, I think allowing students to visually see a process increases their understanding (ex. seeing a

volcano erupt helps them to understand the process involved)

Yes, I would use it to introduce a lesson for the students or have the students use it to create a final

project

Sure. Some simple simulations this works well

Yes, I wouldn't have to create the simulations. Maybe students could also use this program to explain

events in books, grammar usages, and or to tell stories.

I do not feel very comfortable at this time in using it. Maybe after I worked with it more then I would use it.

I could explain something that had already been created though.

If I had more time to study the program and felt more confident about the capabilities

Comments about SimBuilder

Yes, I would use this tool however, I would suggest having training with an experienced user.

Yes, if I was teaching an earth science class. This would proved extremely useful for students who

have poor rote memories but have excellent visual/spatial-induced memories

I would be willing if I had the proper support technician or person that I could call to ask questions.

With this software you really have to know it well and be about to teach others not to be frustrated with

the activity if they do not get it on the first tries.

Yes

Yes, in science of history

Sure, this would be good for young students, since many of them are computer- and video-game-savvy.

This would be a good tool for them to use to understand basic science principles.

What ideas do you have for simulations that would be useful for you or someone in your discipline?

Comments about AgentSheets

Migration patterns, government evolution and development, timelines for various subjects: etc.

Use a simulation on a PowerPoint presentation or have children design simulations of what they think

will happen

None at present (not yet teaching)

Nitrogen cycles, water cycles, metabolism, growth, respiration, photosynthesis.

Science experiments would work well,

None at this time.

Photosynthesis process for plants, volcano erupting, earthquake destruction, erosion, oceanography,

Chemistry (atoms, elements etc)

Growth of cities, changes in voting patterns, other social studies related activities.

Butterfly migrations

Recreating scenes in plays and novels, creating grammar lessons, giving opportunity for students to

create stories.

Having some type of simulations that would work besides the sciences.

Seasonal changes weather changes precipitation

Comments about SimBuilder

I think this is a great way to simulate science material. The program is perfect to illustrate Reactions of the interaction between objects.

This simulation program would serve useful in explaining military history in social studies

classes--especially military events that have been influenced by the weather.

N/A

Keep real life situations

War simulations for world history... geography simulations

Science for grade schoolers, maybe even simulations for older students involving more advanced science (physics, chemistry).

Can you think of any changes or enhancements to this system, especially ones that would make it more useful in creating simulations for novices? Please briefly describe the features that you think are needed in building simulation software.

Comments about AgentSheets

No.

Maybe instead of using the word "agent" for the folders, using "New Object" or something agent is confusing to normal people

Provide easy-to-understand, step-by-step instructions. Include visual aids & graphics

It needs to be easy for the user. Have lots of pre-made simulations to borrow behavior from.

I'm not sure

Have clipart types drawing available so people would not feel they have to draw.

Multi-step process that involved many menus may be difficult for novices perhaps, everything could

be incorporated in one or two menus or even just one large worksheet

It seemed pretty easy for me once I started working on it.

Enhance the tools and have more animation tools

More drawing flexibility, bigger agents. Sound (I don't know if it has is). Text in the simulation.

Be aware that people learn in different ways. Some can be given directions, read through them, and then produce what is there. On the other hand there are others who prefer a more interactive environment.

I would allow more room for adjusting the agents appearance

Comments about SimBuilder

I only think that more detailed instruction on how to work each function would facilitate the user.

The necessary features for building simulation software would be the ability to create objects (paintbrush) and the accessibility to rule making in order to create the movement.

The main features that need improvement is the menu diagram as it proved too complicated and more simplified instructions.

Screenshots for what people will see and how they would use each tool for each activity and object would be useful.

n/a

More system help notes

Any final comments about your experiment activities or the software?

Comments about AgentSheets

Overall, it was fun, however, it highlighted how far software governing simulations have to go in order to recreate the visions that lurk in the human mind. Thanks for the experience.

Overall a great idea - just a little tweaking & it will be great :)

No

Automatic saving of each newly modified simulation so that when you reset it, it doesn't mess up.

It was a fun experience that I would like to use with students in the classroom

I thought that this was a useful tool that needs improvement. If there can be some kind of internet base of section added to this to incorporate the internet into the use of the software--teachers could create simulations using graphics from the internet.

Students would enjoy creating simulations, creating simulations also involves using problem solving skills and exposure and usage of the computer

Good Luck with your project

It was pretty frustrating getting started but its on the right track

Thank you! Although I do not have much computer experience, with a little trial and error and practice, I could use this more fluently. It is a nice program, especially for middle school and elementary students. I think high schoolers could use it too. It would be a nice change of pace for the students.

Thanks for letting me participate, sorry if there were any problems!

Comments about SimBuilder

I know that I would not have been able to complete the assignments if I had not had someone guiding me through the steps for the first assignments. After practicing though, I was able to create a simulation on my own rather quickly.

This simulation software has great potential if changes in the menu programming and instructions take place.

This was a very challenging experiment for me.

n/a

It was fun!

Sometimes using the SW was frustrating and seemed a bit difficult, but w/ a little time, I began to understand how to use the system, despite the learning curve, I believe it's a good tool for science education.

Data Analysis Performance Timings with Excel**Table Anova.0.1**

Anova: Single Factor Learning Time

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
Column 1	11	385.62	35.05636	62.33935
Column 2	7	177.04	25.29143	4.968014

ANOVA

<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	407.903	1	407.903	9.991478	0.006055	4.493998
Within Groups	653.2015	16	40.8251			
Total	1061.105	17				

Table Anova.0.2

Anova: Single Factor

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
Volcano	11	329.12	29.92	118.0466
G2V	7	165.39	23.62714	300.8648

ANOVA

<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	169.4002	1	169.4002	0.907809	0.35487	4.493998
Within Groups	2985.655	16	186.6034			
Total	3155.055	17				

Table Anova.0.3

Anova: Single Factor

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
TLT	11	714.74	64.97636	231.678
G2 TLT	7	342.43	48.91857	312.0911

ANOVA

<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	1103.037	1	1103.037	4.21275	0.056853	4.493998
Within Groups	4189.326	16	261.8329			
Total	5292.363	17				

Table Anova.0.4

Anova: Single Factor

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
Reuse1	10	242.12	24.212	22.23471
G2 R1	6	123.61	20.60167	111.3011

ANOVA

<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	48.8794	1	48.8794	0.904435	0.357726	4.600111
Within Groups	756.6178	14	54.04413			
Total	805.4972	15				

Table Anova.0.5

Anova: Single Factor

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
R2	10	229.55	22.955	26.28785
G2 R2	6	107.4322	17.90537	185.5936

ANOVA

<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	95.62035	1	95.62035	1.149521	0.301784	4.600111
Within Groups	1164.559	14	83.18277			
Total	1260.179	15				

Table Anova.0.6

Anova: Single Factor

Total 12978.76 19

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
TRT	10	471.67	47.167	65.67653
G2 TRT	6	231.0422	38.50704	546.3077

ANOVA

<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	281.2311	1	281.2311	1.184976	0.29473	4.600111
Within Groups	3322.627	14	237.3305			
Total	3603.858	15				

Table Anova.0.7

Anova: Single Factor

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
Total Time	11	1212.53	110.23	218.6911
G2TT	7	573.4722	81.9246	1161.272

ANOVA

<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	3427.336	1	3427.336	5.990184	0.026305	4.493998
Within Groups	9154.541	16	572.1588			
Total	12581.88	17				

Appendix K: Retrospective Interview Questions



Visual Programming Spring 03

Interview Questions

Were there any questions in the Questionnaire you did not understand?

1. What other ideas do you have for simulations?
2. What was more enjoyable drawing or making the simulations work?
3. Do you feel that your students will be motivated to use this environment?
4. What support would you need to utilize this software as a curricula aid for one class (e.g. hardware, training, etc.)?

Any final comments about your experience today?



Visual Programming Usability Testing (SimBuilder)

Name: _____

Date: _____

SSN: _____

I participated in the SimBuilder Usability Test and was compensated the amount of \$15.00 for my work in this experiment.

Signature: _____

Appendix L: AgentSheets Simulation Creation Timing, Objects and Rules Created
AgentSheets Research Results Fall 2002

	Agents	Rules	WC	Vol	L1 Time	R1 Time	R2 Time	Total Time
P1jt	Volcano Starter-Ocean Ocean Sand Ozone> Photosynthesis Flower Sun	 If see a Ocean wave Then move East If see sand at this point, Then move West If the flower sees itself, Then the flower will move upward. If see a Sun here, Then move SE Sun does not work properly.	29:33	17:32	47:10	22:13	33:05	2:00:28
P2kf	Volcano Volcano Steam Lava Ozone> Photosynthesis Plantsandtrees Sky Clouds Sun Reused Starter> Ocean Ocean Beach Clouds Sun Reused	There is a basic portrait of a Volcano and the volcano creates smoke and the Smoke floats up. If See a Volcano, Then create New Smoke above Volcano And Move West And Move East Move North No Behavior Just a landscape no working parts to simulation. No Behavior No Behavior No Behavior No Behavior The simulation works and appears to begin with clouds that move and an ocean that sweeps across the beach. If See an Ocean at this point, Then Move East No Behavior If See a Cloud at this	46:44	32:25	1:18: 25	22:34	20:02	2:10:09

		point, Then Move East No Behavior						
P3km	Volcano Overview	This volcano has several agents layered on top of one another (a mountain body, filled with magma, and covered by a mountain top. The Lava move upward and blow the top off the mountain.	45:08	54:02	1:42:00	18:02	20:07	2:21:07
	Mountainbody	No behavior						
	Mountaintop	If See Sky above, Then Move North						
	Mountaintopbits	No behavior						
	Sky	No behavior						
	Lightening	No behavior						
	Magma	No behavior						
	Magma2	If See Sky above, do nothing						
	Lava	If See self, Then Move North						
	Sky2	No behavior Repeated agent						
	Starter> Photosynthesis	The sun produces sunrays that shine downward and the plants move up.						
	Sun	If See self, Then Create New Sunray to the East						
	Sunrays	If See self, Then Move South East						
	Plant	If See self, Then Move North						
	Ozone> Ocean	Scene starts as an ocean with waves coming out and hitting the beach.						
	Sand	No behavior						
	Waves	If See self, Then Move West						
	Ozone as Ocean	If See Sand, Then Erase If See Sand, Then Change to Weaker Ozone, which may symbolize less water after hitting the sand.						
P4an	Volcano	There is a basic portrait of a Volcano. Lave erupts from Volcanoes and rises into the sky. Clouds also	31:22	23:22	56:01	22:05	14:46	1:32:46

	<p>Volcano</p> <p>Sky</p> <p>Lava</p> <p>Cloud</p> <p>Grass</p> <p>No reuse>Ocean</p> <p>Ocean</p> <p>Beach</p> <p>Waves</p> <p>No reuse >Photosynthesis</p> <p>Sun</p> <p>Sunrays</p> <p>Plant</p> <p>Grass</p> <p>Sky</p> <p>Clouds</p>	<p>float off Eastward.</p> <p>If See a Volcano, Then Create New Lava above Volcano</p> <p>No behavior</p> <p>If See Lava, Then Move North</p> <p>If See a Cloud, Then Move East</p> <p>No behavior</p> <p>There is a basic portrait of an Ocean and Beach. The waves and the Ocean both Move to the East and sweep over the Beach.</p> <p>If See a Ocean, Then Move East</p> <p>No behavior</p> <p>If See a Wave, Then Move East</p> <p>There is a portrait of a Landscape. The Sun is in the sky and plants are on the grass. Sunrays fall southeast from the sky and clouds float eastward.</p> <p>No behavior</p> <p>If See a Sunray, Then Move Southeast</p> <p>No behavior</p> <p>No behavior</p> <p>No behavior</p> <p>If See a Cloud, Then Move East</p>						
P5eh	<p>Volcano</p> <p>Mountain</p> <p>Lava</p> <p>Birds</p> <p>Starter>Photo</p>	<p>The scene begins with a nice portrait of volcanoes that turn into lava, then the lava rising and birds escaping the scene.</p> <p>If See a Mountain, Then Change to Lava And Move North</p> <p>No behavior</p> <p>If See Birds, Then Move South</p> <p>The scene begins with a nice portrait of a sunny</p>	46:05	17:36	1:03:41	17:56	18:07	1:38:32

	<p>Sun Tree</p> <p>Tree2 Sunrays</p> <p>Ozone>Ocean (Uses Starter)</p> <p>Emitter</p> <p>Waves</p> <p>Sand</p>	<p>day with lots of trees. When the little tree sees the sun, it turns into a big tree.</p> <p>No behavior If See a Sun, Then Change to a Big Tree2</p> <p>No behavior If See a Sun, Then Move randomly</p> <p>The scene begins with a basic portrait of a ocean emitter, waves and sand. The ocean emitter creates waves and the sand move to the east.</p> <p>If See an Emitter, Then Create waves to East</p> <p>If See a Wave, Then Move East</p> <p>If See Sand, Then Move Northeast</p>						
P6dc	<p>Volcano</p> <p>Volcano</p> <p>Lava</p> <p>Clouds</p> <p>Land</p> <p>Ozone> Photo</p> <p>Sun (Reused) Solorenergy</p> <p>Land, Land2</p> <p>Starter>Ocean</p>	<p>The scene begins with a basic portrait of volcanoes that spew lava.</p> <p>If See a Volcano, Then Create New Lava above</p> <p>No behavior</p> <p>No behavior</p> <p>No behavior</p> <p>The scene begins with a basic landscape with sun, solar energy and small flowers that grow to large flowers with solar energy.</p> <p>No behavior If See self, Then Move down</p> <p>If Land Sees SolarEnergy above, Then Change to Land2Move</p> <p>The scene begins with a basic ocean portrait with ocean, sand and waves. The sand dissolves in a 3 step animation when it sees waves and the waves wash over everything.</p>	29:41	28:34	57:15	25:03	22:38	1:46:05

	<p>Ocean Ocean1 or waves</p> <p>Sand, Sand2, Sand1</p>	<p>No behavior If See self, Then Move East</p> <p>If See waves to the East, Then Change to a smaller sand</p>						
P7jm	<p>Volcano</p> <p>Ground Volcano</p> <p>Lava</p> <p>Sun Sky</p> <p>Starter>Photo</p> <p>Sun</p> <p>Plant, Bigger</p> <p>Mover</p> <p>Random Mover</p> <p>Ozone>Ocean</p>	<p>The scene begins with a basic portrait of volcanoes that spew lava.</p> <p>No Behavior If (no condition) or Always, Then Create New Lava North, Create New Lava East, Create New Lava NW</p> <p>If (no condition) or Always, Then Move East, Move NE, Move NW</p> <p>No Behavior No Behavior</p> <p>The scene begins with a basic portrait of the sun, solar energy that go through 2 stages moving down and then moving randomly until they come in contact with plants and turns them into bigger plants.</p> <p>If Sun, Then Create New solarenergy (movers) to east</p> <p>If See solarenergy to the west, Then Change to Bigger Plant</p> <p>If See self, Then Move East</p> <p>If 20% change See self, Then Move East</p> <p>If 20% change See self, Then Move Northeast</p> <p>If 20% change See self, Then Move Southeast</p> <p>If 20% change See self, Then Move North</p> <p>If 20% change See self, Then Move South</p> <p>The scene begins with a basic ocean portrait with ocean, sand and waves. The sand is removed</p>	39:00	31:23	1:11:00			1:36:30

	Waves Ocean Beach	when waves hit it. If See sand to the East, Then Erase it If See wave here, Then Move East No Behavior No Behavior						
P8jlm	Volcano Volcano Smoke Lava Pressure Sky Starter>Photo Plant Sun Sunrays Ozone>Ocean Ocean Waves Beach Sand	No Behavior No Behavior No Behavior No Behavior No Behavior A basic sunny landscape with flowers and sunrays created by the sun. Once the sunrays contact the flowers they grow bigger. If See a Sunray above, Then Change to a bigger flower If See a Sun here, Then Create New Sunrays below If See Sunray here, Then Move Down I f See an Ocean here, Then Change to Waves above If See Waves here, Then Change to Beach here If See Beach here, Then Change to Sand here If See Sand here, Then Erase						
P9dm	Volcano	The scene begins with a nice portrait of a volcano sitting on a lake of water. The volcano begins to erupt and puffs of smoke billow up into the air and lava fall down to either side of the Mountain. Good layering of agents they all fit in one square, Mountain on	32:00	22:59	56:00	24:04	27:03	1:51:03

	<p>lake, and lava on tip of mountain.</p> <p>Mountain Water Lava Smoke</p> <p>Starter>Ocean</p> <p>Ocean Eraser (Sand)</p> <p>Ozone>Photo</p> <p>Sun (Reused)</p> <p>Chemicals>Sunr ays Factory >Ground Plant</p>	<p>If Mountain Then Create Lava in this position And Create Smoke above No behavior If 50%, Then Move SE If 50%, Then Move SW If Smoke, Then Move North</p> <p>This is a basic portrait ocean and sand, where the sand absorbs the ocean. (It seems that they meant for the sand to be erased when it was next to the ocean.)</p> <p>If Ocean, Then Move East If See Ocean to the West, Then Erase to the West If See Ocean to the East, Then Erase to the East</p> <p>This is a basic landscape. When the sun shines, sunrays fall to the ground and BRCL elements from Ozone are created.</p> <p>If Sun, Then Create Sunray at position SW Next to >0 Suns, Erase and create BRCL (rule left from chem.) If See Sunray, Then Move SW No behavior</p>						
P10t m	<p>Volcano</p> <p>Volcano Background Ash Ground</p>	<p>The scene begins with a nice portrait of volcanoes that quake and are erupting lava and ash that fills the air.</p> <p>If See a Volcano, Then Move East, Move West No behavior If See Ashes, Then Move North, Move NW, Move NE No behavior</p>	28:36	39:22	59:22	32:18	24:09	1:40:23

	<p>Lava</p> <p>Ocean>Starter</p> <p>Ocean Waves</p> <p>Sand</p> <p>Smsand</p> <p>Photo>Ozone</p> <p>Sun (Reused) Sunrays</p> <p>Smalltree</p> <p>Bigtree Ground</p>	<p>If See Lava, Then Move West, Move SW</p> <p>The scene begins with a nice ocean portrait of with waves and sand. The waves begin rolling out of the ocean and erode the sand to smaller sand.</p> <p>No Behavior</p> <p>If See self, Then Move East</p> <p>If See wave to the West, Then Change to Smsand</p> <p>No Behavior</p> <p>The scene begins with a nice landscape with trees, sun and sunrays. The sunrays fall from the sky and when they hit small trees they will grow to Big trees.</p> <p>No behavior</p> <p>If sunray, Then Move Down</p> <p>If see sunray Above, Then Change to Big Tree</p> <p>No behavior</p> <p>No behavior</p>						
P11p h	<p>Volcano</p> <p>Sky</p> <p>Grass</p> <p>Mountain</p> <p>Sparks</p> <p>Lava</p>	<p>The scene begins with a nice portrait of volcanoes that erupt with lava, and the lava rises into the air.</p> <p>No Behavior</p> <p>If See the Sky, Then Change to less grass</p> <p>If See a Mountain once every 3.0 seconds, Then Create New Sparks above</p> <p>If 33% , Then Move North</p> <p>If 33% , Then Move NE</p> <p>If 33% , Then Move NW</p> <p>No Behavior</p>	33:29	38:03	1:11:32	29:25	25:48	2:07:48
P12lt s	<p>Volcano</p> <p>Volcano</p>	<p>The Volcano is a scene that includes Volcanoes that create sparks, and Clouds.</p> <p>If See a Volcano, Then Create new Sparks above</p>	24:00	23:49	51:49	28:27	23:48	1:43:01

Clouds	No behavior						
Sparks	No behavior						
Ozone > Ocean	A nice oceanscape, with ocean, waves, sand and sun. The waves flow out of the ocean and push the sand down.						
Sun (Reused)	No behavior						
Sand	If See a wave to the west, Then Move Down						
Ocean	No Behavior						
Waves	If See at this point a wave, Then Move East						
Starter>Photo	This is a scene of a sunny day with volcanoes and sparks. Sparks are created by Sun, Sparks move down and volcano erases and creates new sparks.						
Mover (Sun)	If See a Mover, Then Create New Spark Below						
Emitter (Sparks)	If See a Spark, Then Move Down						
Eraser(volcano)	If See a Spark, Then Create New Spark above.						

Appendix M: SimBuilder Simulation Creation Timing, Objects and Rules Created

The following tables represent ANOVA Tables of Simulation Creation Timing.

Table 0.8

Anova: Single Factor Learning Time

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
Column 1	11	385.62	35.05636	62.33935
Column 2	7	177.04	25.29143	4.968014

ANOVA

<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	407.903	1	407.903	9.991478	0.006055	4.493998
Within Groups	653.2015	16	40.8251			
Total	1061.105	17				

Anova: Single Factor

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
Volcano	11	329.12	29.92	118.0466
G2V	7	165.39	23.62714	300.8648

ANOVA

<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	169.4002	1	169.4002	0.907809	0.35487	4.493998
Within Groups	2985.655	16	186.6034			
Total	3155.055	17				

Anova: Single Factor

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
TLT	11	714.74	64.97636	231.678
G2 TLT	7	342.43	48.91857	312.0911

ANOVA

<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	1103.037	1	1103.037	4.21275	0.056853	4.493998
Within Groups	4189.326	16	261.8329			
Total	5292.363	17				

Anova: Single Factor

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
Reuse1	10	242.12	24.212	22.23471
G2 R1	6	123.61	20.60167	111.3011

ANOVA

<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	48.8794	1	48.8794	0.904435	0.357726	4.600111
Within Groups	756.6178	14	54.04413			
Total	805.4972	15				

Table 0.9

Anova: Single Factor

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
R2	10	229.55	22.955	26.28785
G2 R2	6	107.4322	17.90537	185.5936

ANOVA

<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	95.62035	1	95.62035	1.149521	0.301784	4.600111
Within Groups	1164.559	14	83.18277			
Total	1260.179	15				

Table 0.10

Anova: Single Factor

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
R2	10	229.55	22.955	26.28785
G2 R2	6	107.4322	17.90537	185.5936

ANOVA

<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	95.62035	1	95.62035	1.149521	0.301784	4.600111
Within Groups	1164.559	14	83.18277			
Total	1260.179	15				

Appendix N: SimBuilder Session Transcripts

SB04170301

P1

Evaluator Monologue

We practice something that we call “think aloud” protocol. So as you are going through your dialog, mention the things that you are going through and the things that you are reading or thinking. Please say them aloud so that we will know some of what you are experiencing.

00:00:00	Start Exp
00:17:00	Open WaterCycle
00:02:04	Stop
02:35	Trying to change the sun. When the user selects the sun is copied
02:57	Error: User looks in viewer to change color User did not understand the middle button click
04:07	User used the change color button to change color
04:27	User used the paint button to repaint sun
05:04	User accepts change
05:27	User looks into sun viewer “sun doesn’t do anything”
06:06	User drags script <i>Error- User drags individual scripts out of viewer</i>
7:37	<i>Error-User drags moving script and in stead on clicking on forward by and incrementing to 10 expands script, but finally increases to 10</i>
9:25	<i>User repeats task and gets correctly Tests cloud and it works properly</i>
12:02	<i>Change direction of cloud to cloud moving down Selects rotate and change arrow to down position</i>
13:30	<i>User tests cloud doesn’t work</i>
13:45	<i>User tests cloud with yellow bang. Cloud rule works</i>
15:20	<i>Drawing the Bird User easily used paint tools, brush, selector, color dropper, makes a nice large brown bird with yellow wings and brown beak</i>
17:17	<i>User complete Bird</i>
17:36	<i>User adds rule for bird and tests movement. Does not work initially,</i>
18:24	<i>User need to add paused to script. Bird Flies</i>
19:42	<i>User uses rotate handle to change direction of bird flight User changes bird to fly to the ground.</i>
20:41	<i>Bird Flies to the ground.</i>
20:58	<i>Bird finally on the ground.</i>
21:08	<i>End Learning Session</i>
23:47	<i>Saves Water Cycle MM</i>

24:31 *Saved*
25:28 *Published*
25:46 *Press Prev to Return to Main Window*
Reading to begin Volcano project
26:15 *Start New Project*
26:47 *Name Project volcano mm*
Opens blank Project
27:26 *Begin Drawing new Objects*
27:46 *Draws Ground, changes to fill and creates Background*
28:15 *User Creates Blue Background*
28:49 *ERROR user creates too many background and has to delete some.*
29:49 *User creates mountain*
30:21 *User creates sparks on same picture as Volcano*
30:45 *Evaluator, reminds user that if you draw them on the same canvas it is considered as the same object. If you want it to do separate activities you have to draw another object*
31:33 *User creates blue cloud object.*
32:25 *User creates yellow spark object.*
33:02 *User selects spark halo and changes forward direction to up*
33:38 *User opens Viewer to add behavior to spark*
34:01 *User adds rule spark forward by 16.*
34:25 *Error- User has problem with second background they created by accident*
34:56 *User changes forward direction of cloud to left*
35:11 *User tests cloud with clock*
Action- Cloud moves to left edge
36:05 *User adds bounce with scrape sound to cloud*
user tests cloud with clock
Error- User has problem with extra background
37:10 *Cloud works*
38:22 *User wants to change direction of Volcano*
User bring up Halo
User activates rotate
!!Amazing the user's background is the same color as the rotate arrow, so she can't see it.!!
38:28 *User changes color of background to blue*
Evaluator helps user to send background to back
40:46 *User selects Misc scripts*
42:40 *User saves Volcano*
User Comment "Once you get the hang of it. It's pretty simple to use."
End Creating Session 1
00:50 *User Break*
43:30 *User begins Training session 2*
45:43 *User enters Starter World*
45:52 *User Presses Go... Starter runs*

- 46:02 *User duplicates emitter by clicking it*
 46:20 *User clicks stop*
User investigates
- 47:29 *User presses Go, Sim runs,*
 48:03 *User Presses Stop, etc*
 49:26 *User identifies Emitter as good candidate for reuse*
 49:46 *User begins to repaint Emitter as Ocean and Waves as background*
Evaluator reminds user that if they interact they must be on a separate canvas.
- 50:47 *User recreates Ocean,*
Uses Paint brush, then bucket fill to complete
Looks at rules
User identifies Mover as good candidate for Wave
- 51:45 *User create Wave, bucket fill*
 52:19 *User finished with wave*
 53:14 *User tests Ocean*
Wave moves to the right
User resets wave
- 54:30 *User identifies Changes as good candidates for changes her mind*
- 55:58 *User identifies Eraser and repaints it as beach uses brush and bucket fill*
- 56:30 *Complete beach*
 56:34 *User retests sim, clears up playground*
User looks for a test condition for wave to move beach
User tests Eraser touches a Eraser
Evaluator shows user how to get a tile of object name
Test Mover2's touchesA Eraser3
Yes Eraser forward by 20
- 01:02:42 *User changes forward direction of Beach*
Comment... Playground is too crowded
User Delete Beach
Evaluator helps user retrieve from trash
Evaluator helps user resize playground
User rearranges worksheet
- 01:06:06 *Tests Sim. Waves Work, but beach Doesn't work*
 01:07:16 *User looks for better test*
User tests Eraser Color See Beach Color
- 01:07:58 *User Selects new test palette*
 01:08:30 *User Selects colors*
 01:09:34 *User resets sim.*
 01:09:42 *User Retests sim. Test Works*
Water creates Waves. Waves hit sand. And sand recedes.
- 01:10:05 *User Publishes Sim*
End Starter Reuse

Back to main Window
01:10:43 *Begin Ozone Reuse*
01:12:08 *User Enters Ozone World*
01:12:10 *User Presses Go*
01:02:13 *System Error-Eval Investigates*
01:22:30 *System Reset*
01:23:03 *User Creates Plant*
01:23:25 *User Creates Ground*
01:24:07 *User Creates Sunrays 3 separate ray objects*
01:24:15 *User Creates 2 Sunrays as another sunray object*
01:25:06 *User creates growing part of plant*
01:25:40 *User adds rules to plant*
Move forward by 5
01:26:02 *User sets misc. downward direction of sunrays*
01:26:46 *User adds rules to sunrays*
Move forward by 5
01:27:43 *Changes rules to Paused*
01:27:57 *Finished Project II*
01:28:35 *Publishes project*
01:28:56 *Publishing complete*

SBDCL2**P2 P3****Evaluator Monologue about “think aloud”, details, etc.**

00:20

00:31 *Open WaterCycle*

00:39 *User Presses Go to Explore Water Cycle*

01:09 *User Presses Stop*

01:14 *User Reads Exploring the Sun*

01:34 *User investigates sun’s Halo*

02:11 *User selects Cloud Sun Viewer*

02:29 *User begins to repaint Sun, uses repaint tool*

02:52 *ERROR user makes multiple copies instead of inspecting cloud*

03:23 *User investigates Sunray*

03:48 *User begins to change color of the sun*

04:11 *User chooses large brush, and makes it too big, then changes color*

05:02 *User clears and begins to draw another sun*

05: *User **error** needs to get fixed, Evaluator clears desktop*

06:43 *Evaluator tell user to press Toss and cleans up desktop of tooo many suns*

09:35 *User back on task. What does sun do? Investigates rule shining*
Error: User drags rules from inside of scripts.

12:46 *User Reading for task Changing the Behavior of Clouds*

14:46 *User Investigates Cloud raining script, but looking at dark cloud is much more complicated*
Evaluator instructs the user to create a new scripts for Moving

16:43 *User tests rule for dark cloud moving to right, works*
User use yellow bang!

17:20 *User selects rotate tool to change forward direction of cloud*
Uses handle

17:43 *User tests cloud and it moves down properly*

18:05 *User begins reading task Creating a bird*

18:12 *User draws bird (blue birds)... One tooo big, erases and redraws*

19:48 *User keeps birds*

19:59 *User investigates rules for birds*

20:26 *User tries to creates script for bird to move forward*

21:15 *User changes name of object to bird*

21:47 *User begins reading Giving the Birds Behaviors*

22:43 *User creates script for bird to move forward*

23:09 *User changes the name of the script to move*

24:12 *User tests bird script, birds fly up*
User tries to change forward script, looking for an up script

25:10 *User selects bird to bring up Halo*

25:12 *User selects rotate and changes forward direction of bird*

25:21 *System error – User clicks playfield rotate and get*
MessageNotUnderstood: translateBy: Dialog & Playfield disappears
Evaluator tries to recover playfield, Not in trash, etc.
Since task was already complete, E prompts user to go to next task

- 28:28 *User back to main window*
User draws depiction of Volcano
- 31:42 *User begins Volcano Task*
- 31:48 *User begins to draw Volcano*
- 32:30 *User experimenting with volcano, adding lava, and sparks to same picture*
Evaluator reminds P that if things are to interact they must be separate
Objects
- 33:36 *User thinks of next objects needed*
- 34:05 *User begins drawing sparks object*
- 34:44 *Sparks complete*
- 35:01 *User contemplates interactions to add to volcano*
User wants sparks to fly up
- 35:39 *User checks forward direction of sparks*
Selects rotate and checks arrow
- 36:23 *User selects Viewer handle and selects forward by 5 rule for sparks*
Changes status of script to paused
User tests Volcano and sparks fly off top of volcano
- 37:13 *Break*
- 39:13 *Reading New Material Reusing Ozone to make Ocean*
Mismatch between paper tutorial and on screen tutorial
- 39:58 *Brainstorming New Simulation*
- 42:00 *Evaluator reminds user to save and publish Volcano*
- 43:50 *User Opens Ozone World*
- 44:12 *User Presses Go to watch Ozone world*
- 44:47 *User Presses Stop*
- 45:10 *Evaluator and user get fresh etoy for user to create in*
- 48:51 *End of Error section*
- 49:10 *User begins drawing new player Tree*
- 50:19 *Keeps tree*
- 50:46 *User investigates Tree viewer and Renames First Tree*
- 51:38 *User brings up First Tree halo and Duplicates Tree*
- 51:52 *User makes tree duplicate bigger*
- 52:24 *User changes name of big tree to Second Tree*
- 53:55 *User drawing energy*
- 54:38 *User naming energy*
- 55:02 *User Opening viewer for energy*
Double click
- 55:26 *User add script energy forward by 6*
Changes state to paused
- 56:01 *User trying to select energy and rotate and clicks create tile instead*
User had to figure out how to remove tile
User tests energy and it moves upward
- 57:40 *User investigates viewer to find a good rule for energy*
User wants energy to bounce
- 58:12 *User adds energy bounce and make croaking sound*
Error User drags rules out of scripting tile

Evaluator reminds user that scripts don't work if not in script
User tests rule
User selects energy to change forward direction
59:31 *User publishes and saves project photosynthesis*
60:13 *Publishing complete*
60:22 *User back to main screen*
User reading for next task
Error enters worlds of Squeak
60:57 *User enters starter world*
61:06 *User Presses Go to begin starter world*
User Runs a couple of times to investigate process
62:43 *User begins to read specs for Ocean world*
64:20 *Users cleaning up workspace*
65:31 *User begins to create new players*
User begins with cloud and then
Changes mind and wants to use mover as cloud
66:31 *User begins to repaint mover into cloud*
User creates 3 big clouds as if it were background.
68:18 *P erases cloud and draws one big cloud*
68:41 *User duplicates cloud and resizes this cloud to be smallcloud*
and renames small cloud
69:20 *System error*
70:01 *recovered from Sys error*
Nope
Randommovers are running wild
70:31 *Delete wild random movers...*
71:25 *User Creating player Ocean*
User using small pen
User selects bigger pen to work faster
72:25 *User Keeps and names ocean*
Error User clicks on Playfield tile instead of sketch tile
Finally gets ocean tile and renames
73:17 *User begins land object*
User selects large brush to create land with muddy divide and creates a
tree on the land
75:01
76:05 *User changes forward direction of ocean*
User resizes etoy to have more room
User resizes playfield to have more room
When Ocean see land it will change it's heading in the opposite direction
79:01 *User adds rule ocean forward by 5 and tests simulation*
80:03 *User saves project as Oceanerosion*
80:38 *Project published*

SBKR4**Evaluator Monologue**

We practice something that we call “think aloud” protocol. So as you are going through your session, mention the things that you are going through and the things that you are reading or thinking. Please say them aloud so that we will know some of what you are experiencing.

User: So just talk out loud

Evaluator: Yes

Use the center mouse button to get halo

P4

00:00:00 Start Exp
 00:00:04 Open WaterCycle
 00:00:07 User Runs WaterCycle
 User selects
 00:00:39 User selects cloud and uses flap to close
 00:00:56 User selects repaint and changes color of sun
 00:00:59 System error
 00:04:59 System error corrected
 00:05:52 User beginning to change cloud
 00:06:52 P4 reads *Changing the behavior of the cloud*
 00:07:06 *Looks at rules for clouds*
 00:09:27 *User changes the*
 00:09:56 *I'm going to make it move vertical now*
Evaluator: User does not see directions and experiments with moving
And doesn't know where to find the down arrow.
User: Selects rotate and move it a tiny bit
I'm going to point it down
 00:11:37 *One of the clouds is going all the way down to the ground*
Drawing the Bird
 12:17 *I'm going to select a brush and a color and go over here*
I don't like that bird. User throws away and starts again
I haven't drawn my bird yet
 14:20 **User draws a pink bird, user throws away and user draws a red bird**
 14:46 *So I draw one bird and press Keep*
 14:53 *OK. Now we want it to be able to fly*
I have my bird here and ...
What do we want the bird to do
 15:20 *I want it to make a sound*
Evaluator: Click on the basic set of rule and there is something that will
make a sound
I want it to move forward and make a sound
Evaluator: Can you hear it?
User: No
Evaluator: You may have to turn it up to hear the sound.

- 17:22 *P4's bird chirps. E: You should combine the rule in one*
 17:36 User adds name of rule.
 User complete Bird
 System Error
- 18:10 User adds rule for bird and tests movement.
 19:21 Error System flap in way
 Error changing name too close to top of window
 User creates a bird that chirps and flies
 User easily adds rules to script and drags unwanted ones to trash can.
- End Learning Session**
- 25:22 *User Begins reading to **Create a Volcano***
 User draws depiction
- 27:45 User start New Project
 User selects new project
- 28:30 Rename and Enters *blank Project*
 29:00 *The first thing I'm going to draw is my **volcano***
 User has no problem with drawing tools. Doesn't like the first drawing
 Toss's it and begins another
P4 I drew the volcano. Now I need to draw some of the things around it.
So I'm going to go back, get a paintbrush and get a color.
- 30:16 *User begins drawing an ocean around Volcano. **Layering***
 30:33 *P4 And basically what I'm making now is my ocean.*
 30:47 *E Doesn't like it so user hits clear and begins again.*
*P4 But I did mean to do that... so I'll hit **Clear** and I'll go back and try*
this again.
Drawing Error
- 31:38 **E User using paint bucket to fill the lake but it fills the entire space.**
P4 presses Undo and Tries it again a couple of times
E You must have a hole somewhere in your drawing.
P4 There can't be any holes
E No There can't be any holes P4. OK
- 32:03 *P4 Alright so there's my water, but I'm not happy with it so I'm going to*
add a little bit more. It might take a long time, because I'm not happy
with what I'm doing.... I'll get my bucket. So we have water
- 32:28 *P4 So now we have our ocean*
 32:29 *P4 So what I'm going to do is to add some bird. For some odd reason I*
like birds.
E Probably Just put one. So after you create one you can just copy the
Rest. You have to keep it.
- 33:02 User keeps bird
 33:12 User creates grass **Layering**
P4 I'm creating my grass which is at the bottom of the volcano and I'm
trying to remember to make sure it has a closing area... so the paint won't
go all over the screen... Now I have grass and one bird and press Keep.
- 34:08 *P4 This time I will do a cloud. I can't do it with no color whatsoever so*
I'll do it with grey. I'll make a grey cloud. That's not really what I want

so I will press undo. Go get the paintbrush. I'll have an aqua looking cloud then. We have one cloud.

- 35:15 **P4** *Now let's have an eruption.*
I'm learning that all I have to do is trace a little something out and use the paint bucket to fill it in. Let's add some yellow streaks in there and keep it
- 36:43 *Let's go back and add Lava*
So I must have a hole in it somewhere.
 User uses fill and it fills entire area about 2-3 times.
- 37:55 Lava complete
- 38:09 User adds rain
P4 *The more things I add the more complicated it will be.*
- 38:51 User adds big black boulders and changes them to smaller orange ones.
- 39:13 *Ok I'm done.*
E Now with the things that you have here. Can you think of what you want them to do. P4 Yes I can think of what I want them to do.
Ummm wait I want to add one more thin
Because if you were teaching a class...
E The user adds a symbol
P4 *This is pressure. You would Let the class know that the pressure starts in the ocean and goes up. Now I would need to get them to interact.*
Just think of any object and what you would like it to do.
 User adds the name of some objects.
 User has trouble selecting bird. When they drew bird a lot of things on the same canvas. User is creating a new bird and names it bird.
- 45:44 **P4** adds rules Birdie forward by 10 and Bird make sound warble

P4 *I would just like the rocks go up.*
E reminds the user to click the eye to see the rules.
P4 adds rule Rock forward by 10
- 46:42 **P4** *My water I just want it to shake and make the sound splash*
- 47:00 **P4** *My arrow. I want it to go forward by 3*
P4? *Did I do that for the ocean*
 User checks rule for ocean. It's ok
- 48:37 User want to add motion to eruption and cloud
- 48:56 User test cloud and users rotate to change direction of movement
 User checks rain
- 49:38 **P4** add name to rain and changes direction to down
- 51:05 **P4** publishes project and runs project
User saves Volcano
E: I need to have a reset button
- break
- 56:50 *User begins reading for reuse project*
Begins drawing Ocean world
We will have sand a beach and a bird flying, cloud and sun. That's it.

- P4 Reads... The starter world contains*
- 58:10 *User opens Starter world*
System Error (Eval had wrong starter)
E Can you think of any similarities between something in the starter world and things in your ocean world
P4 The arrow would be like a wave.
P4 creates a wave by opening mover and clearing it and creating wave.
P4 the emitter would be like the ocean.
E so you would need to draw it.
- 64:04 *User draws ocean. Initially sketches and fills whole space, clears, redraws and uses fill. P4's Ocean works to create a mover.*
User's attempt works but because of system it produces an old mover not the new mover. Mover's copy must align after for it to work properly.
- 68:41 *P4: And maybe some sand and that's it.*
System error too many movers in the way and no big eraser)
- 69:17 *User continues to make beach*
E: Can you think of anything else you want to happen other than the ocean creating waves and the waves hitting the beach.
- 70:27 *P4 That's good enough for right now.*
- 70:58 *P4 Publishes project*
-
- 71:58 *User begins Ozone -> photosynthesis*
User runs sim
- 72:24 *P4 So these are coming from the factory to deplete the ozone layer.*
If I were to think of photosynthesis
I would basically have grass and a flower and then sunlight and sunrays coming down.
The rays would come down and shine on the flower
P4 Sun would be emitting sunrays
E You would delete that one because it doesn't have the behavior you want
- 75:04 *User draws a sun to reuse behavior of smoke stack.*
It makes a chemical copy after .
Takes chemicals and repaints it as sunray.
E Is your sunray doing what you expect?
P4 No I want it to move down. I need to select it and take the arrow and move it down
- 77:25 *System Error too many BrCl in one spot and they hadn't deleted ozone...*
- 78:49 *P4 tests sim again*
Works properly but must use sunray's copy for it to work properly.
System Error dialog
- 01:28:56 *Publishing complete*

SBAL5**Evaluator Monologue**

We practice something that we call “think aloud” protocol. So as you are going through your dialog, mention the things that you are going through and the things that you are reading or thinking. Please say them aloud so that we will know some of what you are experiencing.

00:00 Start Exp
User Reads: Right now I'm reading preliminaries
I don't see an interaction guide so I won't

01:01 Exploring the WC model

01:22 Open WaterCycle
Ohh I hear the rain.

: *Change direction of cloud to cloud moving down*
Selects rotate and change arrow to down position

: ***Drawing the Bird***

17:17 *User complete Bird*

17:36 *User adds rule for bird and tests movement.*
Does not work initially,

18:24 *User need to add paused to script. Bird Flies*

19:42 *User uses rotate handle to change direction of bird flight*
User changes bird to fly to the ground.

20:41 *Bird Flies to the ground.*

20:58 *Bird finally on the ground.*

21:08 ***End Learning Session***

23:47 *Saves Water Cycle MM*

24:31 *Saved*

25:28 *Published*

25:46 *Press Prev to Return to Main Window*
Reading to begin Volcano project

26:15 *Start New Project*

26:47 *Name Project volcano mm*
Opens blank Project

27:26 *Begin Drawing new Objects*

27:46 *Draws Ground, changes to fill and creates Background*

01:27:57 *Finished Project II*

01:28:35 *Publishes project*

01:28:56 *Publishing complete*

SBJB6**Evaluator Monologue about think aloud, etc.**

- 00:20 P6 reads Starting Visual Programming Tutorial
P6 reads Exploring Water Cycle Model
Clouds are moving, raindrops are falling, sunrays falling, in the simulation.
- 00:05 *User begins by repainting the sun with a paintbrush and user finds bucket*
P6 reads What does the sun do?
P6 The Sun shines, but I see some other things but I don't think they apply
E There are a lot of things available but you won't need them all
Select a rule and drag and drop it
The cloud moves
Changing the Behavior of clouds
Make clouds more active
Increase it to 20.
- 14:08 *Ok I have these boxes to appear to the right of the value, do I need to worry about them. Not right now close the cloud.*
- 14:44 *Press exclamation*
Open another cloud. Now click the exclamation point and see what happens.
- 15:50 *As I'm pressing the exclamation mark it tests the rule*
- 16:46
- 17:20 *User testing changes.*
- 17:52 *P6 reads Make the cloud move vertically*
- 18:16 *P6 reads Select rotate and move your cloud a tiny bit*
Error flap in the way
- 18:44 *Halo appears and use tries to grab object in the center and not on the rotate icon*
- 19:49 *User makes change but it doesn't work*
- 23:20 *P6 So I got the cloud moving*
System Error- Moving to slowly
- 24:19 *Creating a Bird*
Select Brush and draw something that looks like a bird
- 26:23 *User presses Keep to finish bird and adds it to playground.*
User has extra blob on bird sketch has to press repaint and erase blob
Bird has been added
- 27:51 *Add behaviors*
Mover move
Evaluator that is the same as bird move
- 28:44 *Move forward by 5 is the same as mover move*
But you must add your new rule to a script
- 29:45 *Press go to see how your bird reacts*
It seems to go up
E change action from normal to paused

- Why is it moving it up?*
Think about the working of the cloud
So it has the same workings as the cloud. E well all objects start with the same default characteristics
User deletes bird by accident
E tell her to recover it from the trash can
E Look back at the page where you changed the cloud and think about that
P6 so you rotate, ok
- 32:26 *I didn't change it to the right I changed it to the left*
E, but that is fine that is the way your bird is flying
P6, so now I need to put the bird on the ground
P6, what do I need to do to make it move...
- 34:14 *The user just drags it to the ground.*
- 35:28 *P6, but the users thinks of changing the rotate to get it to the ground*
- 35:40 *P6, So save it*
E Press Prev to get back
System Error, Slow and bad dialog.
- 41:00 *P6 reads Creating a volcano simulation*
User drawing sim
- 42:10 *P6 Lava, sparks coming out of it*
- 42:28 *P6 reads Press New project at end of navigator and name and enter*
- 44:20 *P6 So we just start drawing... Let's see*
P6 draws grass
- 45:29 *P6 draws sun*
- 46:18 *man that looks awful... (user erases with small eraser) let me try again*
I'm still wondering how to get nice even suns like yours, but I will try this until I figure out how you did it. User tries 3 more times
E you can click toss if you don't like them and use circle template
I don't like the size of it. I like the size of this one. It's funny looking but I will use it.
- 48:01 *P6 Now I need a volcano. P6 draws volcano, adds lava on top, uses bucket to fill, uses multiple size erasers to make changes*
- 50: 30 *P6 draws sparks*
P6 There are supposed to be sparks.
E Do you want the sparks to do anything?
P6 If so they should be drawn differently right.
E: They should be on a separate page or canvas
- 52:12 *P6 says I need to draw sparks and begins to draw sparks*
Ok.
- 54:02 *P6 Now I want them to fly out of the volcano*
User reviewing rules
- 55:12 *P6 saves & publishes sim*
P6 tests simulation
- 57:54 *User needs to change status of the rule from normal to paused*
- 58:00 *End Volcano session*

At the end of learning session. E informed user that they could use the largest circle template to make sun.

- 01:00:30 Exploring Visual Programming Section II
User begins drawing sun, rays, maybe... OK
- 01:01:10 The Ozone depletion simulation
- 01:03:29 User creates a new project
- 01:04: So I rename this photosynthesis
E: No just rename it at the end
Task 2 creating and reusing a player's behavior
Reuse a player by just using the paint brush and change it's look
Task 3 new players. So I'm going to draw it.
- 01:06:02 *P6 So, I need to draw a plant that grows...
And rays that hit the plant.. to make it grow.*
- 01:06:34 *E: Now can you think of anything in this world, like you want your world to work
P6: This building the factory actually works like I want the sun to work. Because the rays would actually come from the sun
E: Well actually the smokestack not the factory is actually working
P6: Can I copy it
E: You can click on it with your middle button and you want the halo to appear around it
E: Click on the top right green icon to duplicate
Actually I want to use this for the sun and the plants
E Oh you have to close that one
P6 So I need to do this
E the icon above it is redraw
The one above the eyedropper*
- 01:08:33 User drawing a sun. *P6 Here we go I'm not worried*
- 01:09: *Now you might want if you want to still with your middle mouse button
And you can delete the old sun or put it in your new window*
- 01:09:48 *P6 I'd better change the behavior
E If you want get a new name tile
The current name is Smoke stack
There is a menu above it
P6 Oh that's ok
Hand me a tile of myself
P6 Ok, Hand me a tile for self... Ok*
- 01:10:54 *P6 I'm going to make the rays. User is drawing lots of sunrays*
- 01:12:38 User completes rays and changes direction to downward
- 01:14:32 *P6 Maybe*
- 01:14:55 *P6 If the sun behaves like my smokestack...
E So far how many new agents do you have? P6 Two
E at does your sun do?
P6 If it works like the smokestack...I want the rays to move away from the Sun like the chemicals move away from the smoke stack.*
- 01:17:55 User names sunrays

01:18:17 *I want you to move down instead of up.*
01:18:51 *User fixes rays*
01:19:09 *User draws flowers*
01:19:58 *Completes flower and places it*
It should be the same behavior as the sun, so lets take the old sun
01:21:54 *Oh I don't like that*
User redraws
01:22:21 *User complete*
01:22:43 *User names flower*
01:23:43 *P6 How to make rays that keep coming out*
01:25:49 *P6 So the rays come to the ground and how do they disappear*
01:27:58 *P6 Ah it's the flower that should ...*
01:32:19 *P6 I'm trying to get the sunrays to turn it into the bigger flower.*
Rays need to have the same behavior as Chemicals
01:33: *I think I need that*
01:35 *I'm trying to figure our how to change*
Any problems
01:37: *I attempted*
The big flower is the BCL, but I don't figure out what happened to it
And I couldn't find it.
E: Maybe it's in the trash
P6: Here it is
You just want that part to get bigger when the rays hit it.
Because when I step through it it doesn't work
It probably did work, but it may have stopped by accident.
You want it to replace with petals instead of BCLs
As long as you can understand how to do it.
I wanted to be here, but it falls off and I need to modify and not move off
but just grow in that place.
We can kinda cheat and align after the dot.
I need to create something that creates in this place. I will created but it
does not create on front.
So it sort of works it makes bigger petals when the sun hits it.
All of you can go away.
01:45:01 *Publish photo JB*
There is a problem with simulation
E: Oh the test is missing; If you add it back it will work
These are all the testing conditions...
You would have to test with the color, and click on the flower color
When it see the rays, so click on that color
01:50:24 *OK now it should work. Now if that is true we want it make those petals.*
P6 if Yes... Ok
E Ok now it works
P6 that's basically what I wanted
01:51:27 *Publish again*

01:54:00 *Begin Starter->Ocean
start a new project*

01:57:26 *User beginning beach sand*

01:58:57 *User completes eroded sand*

01:59:22 *User repaints Replacer as regular sand*

02:00:11 *User copies*

02:01:42 *P6 ...should be the waves*

02:03:30 *How can I change this text
First you create those objects
Because what did you want
I want the wave to be a random mover
Click on the eroded sand
There's a red icon at the top
P6 the menu
Under the eye there is a tab that is the name of it
P6 ok thanks*

02:05:09 *Erased eroded sand
When you click on it to make a copy.*

02:06:02 *P6 Now I can draw the water
User begins drawing water*

02:06:56 *User completes water*

02:07:16 *Need waves*

02:07:34 *User draws waves, uses small eraser to fix look*

02:09:01 *P6 doesn't like so deletes waves and draws more examples
and erases undesired ones with big eraser*

02:10:15 *User trying to remove eroded sand from trash
unsuccessful initially*

02:11:40 *P6 found eroded sand
User is getting it to work
When mover hits the replacer it turns into eroded sand
They don't need to be in the scene yet. So do you need more room.
Ok so You can make the playfield bigger
The waves have the same rules as the mover.*

02:16:59 *The emitter should work like the ocean
You don't need the page you need the actual object.
So that is trash. Click your middle mouse and X it out*

02:18:17 *P6 Where would your waves come from the emitter
which is the ocean
E You don't have to change it. You can drag the rule from the other
Object
P6 Copy the mover rule from the emitter to the ocean
P6 So Let's change these colors
So click on the ocean and
Now it doesn't have any behavior
Now you can grab the whole rule and copy it to the ocean and the mover
in your instance is a what.*

P6 a wave, emitter is the ocean and it has to have status paused
Something happened
The waves hit the sand
P6 now when the waves hit the sand it make eroded sand.
P6 Here is the script
P6 When the waves overlap the initSand I want the eroded sand to copy
align after initSand
P6 Ok...
P6 Initial sand I got that
02:24:26 P6 When the wave hits the sand...
02:25:31 So we will probably have to bring it up a little bit and reposition your rules
You have to select the new color
Test waves is over color brown
 Yes waves_jb delete
 Eroded sand copy align after sand
 No waves forward by 5
P6 Tests sim
Small error
P6 and E look in another object for rule and double checks it

02:28:51 P6 Tests sim
Ocean creates waves and when wave hits sand, it erodes the sand
And makes it smaller
E the rule slipped out the wave is over sand and it works
It just slipped out
02:31:06 I'm trying to figure out exactly how it works and get the art looking right
and just get it back to it's initial state to run
P6 Publishes and saves project.

P5 maybe...

28:15 *User Creates Blue Background*

28:49 *ERROR user creates too many background and has to delete some.*

29:49 *User creates mountain*

30:21 *User creates sparks on same picture as Volcano*

30:45 *Evaluator, reminds user that if you draw them on the same canvas it is considered as the same object. If you want it to do separate activites you have to draw another object*

31:33 *User creates blue cloud object.*

32:25 *User creates yellow spark object.*

33:02 *User selects spark halo and changes forward direction to up*

33:38 *User opens Viewer to add behavior to spark*

34:01 *User adds rule spark forward by 16.*

34:25 *Error- User has problem with second background they created by accident*

34:56 *User changes forward direction of cloud to left*

35:11 *User tests cloud with clock*

Action- Cloud moves to left edge

36:05 *User adds bounce with scrape sound to cloud*

user tests cloud with clock

Error- User has problem with extra background

37:10 *Cloud works*

38:22 *User wants to change direction of Volcano*

User bring up Halo

User activates rotate

!!Amazing the user's background is the same color as the rotate arrow, so she can't see it!!

38:28 *User changes color of background to blue*

Evaluator helps user to send background to back

40:46 *User selects Misc scripts*

42:40 *User saves Volcano*

User Comment "Once you get the hang of it. It's pretty simple to use."

End Creating Session 1

00:50 *User Break*

43:30 *User begins Training session 2*

45:43 *User enters Starter World*

45:52 *User Presses Go... Starter runs*

46:02 *User duplicates emitter by clicking it*

46:20 *User clicks stop*

User investigates

47:29 *User presses Go, Sim runs,*

48:03 *User Presses Stop, etc*

49:26 *User identifies Emitter as good candidate for reuse*

- 49:46 *User begins to repaint Emitter as Ocean and Waves as background
Evaluator reminds user that if they interact they must be on a separate canvas.*
- 50:47 *User recreates Ocean,
Uses Paint brush, then bucket fill to complete
Looks at rules
User identifies Mover as good candidate for Wave*
- 51:45 *User create Wave, bucket fill*
- 52:19 *User finished with wave*
- 53:14 *User tests Ocean
Wave moves to the right
User resets wave*
- 54:30 *User identifies Changes as good candidates for
changes her mind*
- 55:58 *User identifies Eraser and repaints it as beach
uses brush and bucket fill*
- 56:30 *Complete beach*
- 56:34 *User retests sim, clears up playground
User looks for a test condition for wave to move beach
User tests Eraser touches a Eraser
Evaluator shows user how to get a tile of object name
Test Mover2's touchesA Eraser3
Yes Eraser forward by 20*
- 01:02:42 *User changes forward direction of Beach
Comment... Playground is too crowded
User Delete Beach
Evaluator helps user retrieve from trash
Evaluator helps user resize playground
User rearranges worksheet*
- 01:06:06 *Tests Sim. Waves Work, but beach Doesn't work*
- 01:07:16 *User looks for better test
User tests Eraser Color See Beach Color*
- 01:07:58 *User Selects new test palette*
- 01:08:30 *User Selects colors*
- 01:09:34 *User resets sim.*
- 01:09:42 *User Retests sim. Test Works
Water creates Waves. Waves hit sand. And sand recedes.*
- 01:10:05 *User Publishes Sim
End Starter Reuse
Back to main Window*
- 01:10:43 *Begin Ozone Reuse*
- 01:12:08 *User Enters Ozone World*
- 01:12:10 *User Presses Go*
- 01:02:13 *System Error-Eval Investigates*
- 01:22:30 *System Reset*

01:23:03 User Creates Plant
01:23:25 User Creates Ground
01:24:07 User Creates Sunrays 3 separate ray objects
01:24:15 User Creates 2 Sunrays as another sunray object
01:25:06 User creates growing part of plant
01:25:40 User adds rules to plant
Move forward by 5
01:26:02 User sets misc. downward direction of sunrays
01:26:46 User adds rules to sunrays
Move forward by 5
01:27:43 Changes rules to Paused

Appendix O: AgentSheets SimBuilder Artifact Comparison



Expert User Certification

1. Degree/Certification in Computer Science or related field:

2. Please list experience in software engineering or software/instructional design and indicate whether your experience as either student and/or instructor.

3. Please list experience in Usability Engineering, User Interface Design, Interface Evaluation or Graphic Design.

The rationale for this evaluation is to ascertain the aesthetic quality of these artifacts. We are not trying to find out the quality of the artist, but the perceived quality of the artifact (i.e. being a good representation of the desired model). You will be presented with six models for topics relevant to middle school physical science.

- Water Cycle Model
- Pollution or Ozone Depletion Model
- Generic or Starter Model
- Volcano Model
- Photosynthesis Model
- Ocean Model



Center for HCI: User Interface Evaluation Rating Guide



Ratings	Action Representation	
5	Excellent	A rating of Excellent indicates that the user models the phenomena with a variety of relevant working objects (5 or more).
4	Above Average	A rating of Above Average indicates the user has fairly realistic depiction with at least 4 working relevant objects.
3	Average	A rating of Average indicates the user has a realistic depiction with at least 2-3 objects working.
2	Below Average	A rating of Below Average indicates the user has a at least one object working.
1	Poor	A rating of Poor indicates that the user model has no working objects.

Ratings	Visual Representation	
5	Excellent	A rating of Excellent indicates that the user had good realistic drawing that models the phenomena with a good use of color and 5 or more objects.
4	Above Average	A rating of Above Average indicates the user has fair use of color and realistic depiction with at least 4 relevant objects.
3	Average	A rating of Average indicates the user has fair use of color and realistic depiction with at least 3 relevant objects.
2	Below Average	A rating of Below Average indicates the user has a fair use of color and realism with at least 2-3 objects.
1	Poor	A rating of Poor indicates that the user vaguely conveyed the model with 2-4 objects.

The following is an example of an artifact to evaluate: Expert created.



Water Cycle

Visual Representation Rating

Select the response that best indicates your rating.

- 5 Excellent
- 4 Above Average
- 3 Average
- 2 Below Average
- 1 Poor

Water Cycle Actions

Clouds produce rain,

Water evaporates from lake.

The Sun produces sunrays and sunrays dry the grass.

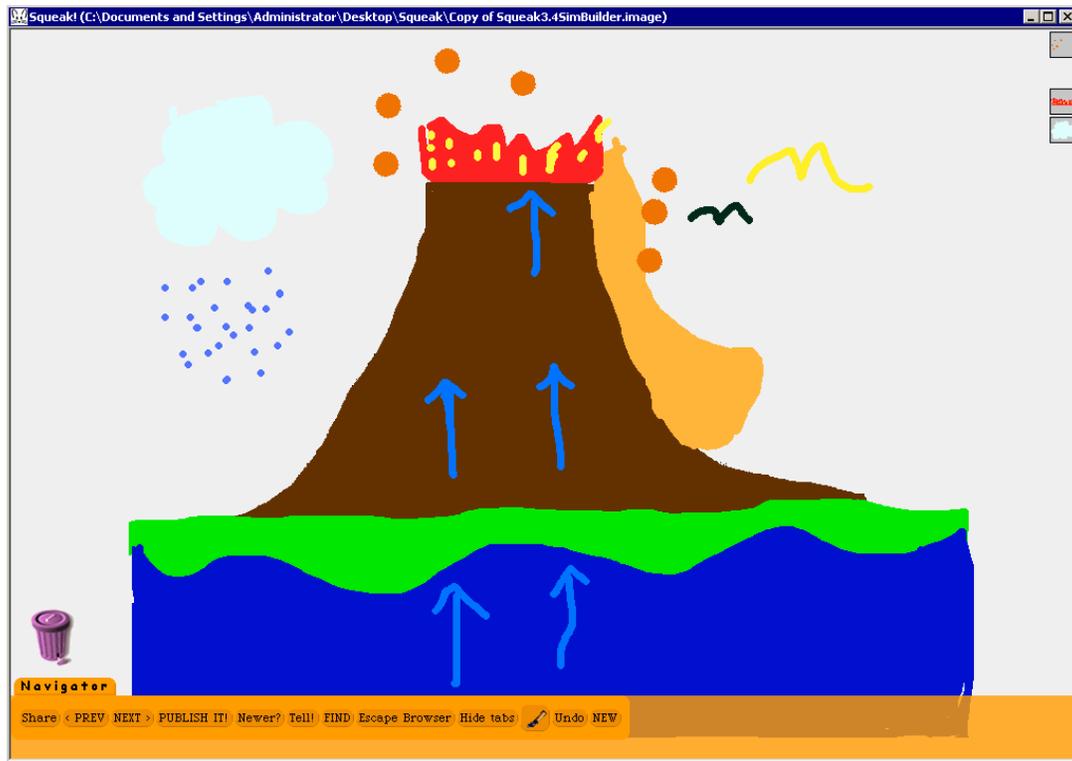
Clouds also move from place to place and sunrays move downward.

Action Rating

Select the response that best indicates your rating.

- 5 Excellent
- 4 Above Average
- 3 Average
- 2 Below Average
- 1 Poor

The following is an example of an artifact to evaluate: End User created.



Volcano

Visual Representation Rating

Select the response that best indicates your rating.

5 Excellent

4 Above Average

3 Average

2 Below Average

1 Poor

Volcano

Volcano with lava, explosions of boulders, ashes, and pressure that rises to cause eruption.

Water and earth surround the Volcano and a bird flies to escape the volcano

Action Rating

Select the response that best indicates your rating.

5 Excellent

4 Above Average

3 Average

2 Below Average

1 Poor

Vitae**CHERYL D. SEALS**
Curriculum Vitae**Business Address:**

Auburn University
Computer Science & Software Eng.
105D Dunstan Hall
(334)844-6319
cseals@vt.edu

Personal Address:

1309 Gatewood Drive
Apt. 910
Auburn, AL 36830
(334)821-7590
<http://csgrad.cs.vt.edu/~seals>

EDUCATION

Ph.D., Computer Science, *Virginia Polytechnic Institute and State University*, August 2004. **Dissertation Title:** A Framework for Reuse in Visual Programming Environments: Supporting Novice Programmer Development of Educational Simulations.

Advisor: Dr. Mary Beth Rosson.

M.S., Computer Science, *Virginia Polytechnic Institute and State University*, May 1997.

M.S., Software Engineering, *North Carolina Agricultural & Technical State University*, May 1995. (Inducted into Upsilon Pi Epsilon Computer Science Honor Society)

B.S., Mathematics Education, *Grambling State University*, Grambling, Louisiana, May 1993.

B.S., Computer Science, *Grambling State University*, Grambling, Louisiana, May 1990. (Inducted into Kappa Delta Pi Education Honor Society)

EXPERIENCE**Assistant Professor**

Auburn University, Auburn, AL
8/2003-present

Research Interests: Human Computer Interaction, User Interface Design, Software Engineering, Mobile Devices, Artificial Intelligence
Teaching: Artificial Intelligence, User Interface Design

Graduate Research Assistant (Usability Engineer)

Virginia Tech, Blacksburg, VA
1995-present

Software/Usability Engineering of visual programming environment to support novice programmer development of educational simulations (2000-present). Usability Analysis of MyInBox Voice e-mail System (1999). Usability Analysis of Digital Libraries (1997-1999), Usability Analysis of Visual Programming Environments (1998-present). Usability Analysis of Scenario Based Tools for Learning Object-Oriented Design (1997).

Graduate Teaching Assistant: CS1004/1104 Computer Literacy

Virginia Tech, Blacksburg, VA

1/2000 – 1/2001

Graduate Teaching Assistant: Aided in Preparation of exams and

management of online testing system, and lab assistant.

Computer Literacy and Programming Instructor

Upward Bound Summer Program, Blacksburg, VA

1997-present

Instructor: Prepared lecture, assignments, exams and delivered computer literacy and programming instruction to Upward Bound High School Summer Students. Topics covered: MS Office proficiency, programming concepts and Web page design and programming.

Graduate Teaching Assistant: CS1044 Introduction to Programming in C

Virginia Tech, Blacksburg, VA

8/95-present

Graded programs in C. Lab monitor and aided students in learning programming concepts.

Application Integrator

IBM (Integrated Services and Solutions Corporation), Atlanta, GA

1995 Health Industry Commercial Contracts - Created proposals and prototypes for health industry contracts as Object Oriented advisor. Developer/tester of Health industry interfaces.

Summer Intern

Polaroid Corporation, Boston, MA

6/94-8/94

Film Imaging Research and Development - Developed a data acquisition application for the Hunter Colorimeter using C Programming and Sapiens SmartStar (Booch, Yourdon, and Fusion Methods)

Database Administrator/System Tester

Bell Communications Research, Piscataway, NJ

1990 - 1992

Member Technical Staff

- Coordinated, executed, and documented the transition of databases from a UNIX to an IBM MVS/IMS environment.
- Increases the efficiency of Product Test Group by saving versions of test scenarios with an automated testing facility.
- Tested UIs, developed automated regression test package to maximize quality and scope of Testing.

COMPUTER SKILLS

Application Programming, System Installation/Monitoring/Coordination, Web Page Design, DataBase Administration. ASTRA, Software Implementation and Documentation.

Courses: Human Factors Research Design, HCI for Collaborative Systems, Computer Supported Collaborative Works, Object Oriented Analysis/Design, Models of HCI, World Wide Web the Underlying Technology, Digital Libraries, Information Storage and Retrieval, Algorithm Analysis/Design, Project Management, Software Analysis and Design, C/C++ Programming, Models and Analysis, Programming Languages, Operating Systems, Technical Writing

Software: Visual Programming Environments, Rational Rose, Sapiens Smartstar, Texas Instruments IEF Case Tool, Visual C++, Enfin Smalltalk, Microsoft Access/Visual Basic, Authorware Professional, Impromptu by Cognos, Window Maker by Blue Sky Corp, RFFlow (for General Design), Xfig (for General Design)

Hardware: DEC, NT, Alpha, and SUN Workstations. IBM, IBM compatibles, and Macintosh PCs

Languages: C/C++, JAVA, HTML, Smalltalk, Scheme, Lisp, Lex, Prolog, Pascal, PL/I, Fortran, LaTeX

PUBLICATIONS

Thesis: A Framework for Learning and Reuse in Visual Programming Environments: Supporting Novice Programmer Development of Educational Simulations.

Committee:

Dr. Mary Beth Rosson, Dept. of Computer Science, Virginia Polytechnic Institute & State University (advisor)

Dr. John Carroll, Depts. of Computer Science and Psychology, Virginia Polytechnic Institute & State University

Dr. Roger Ehrich, Dept. of Computer Science, Virginia Polytechnic Institute & State University

Dr. Rex Hartson, Dept. of Computer Science, Virginia Polytechnic Institute & State University

Dr. John Burton, Department Head, Education Teaching and Learning, Virginia Polytechnic Institute

Refereed Journal Articles:

- Rosson, M. B. and Seals, C. (2000). Learning and Reuse of a Visual Programming Language, IEEE, Visual Languages Journal. September, 2000.
- Kengeri, R., Seals, C., Harley, H., Reddy, H., Fox, E.A. (1999). Usability study of digital libraries: ACM, IEEE-CS, NCSTRL, NDLTD. International Journal of Digital Libraries. August 24, 1999 2: 1-13.

Refereed Conference Papers:

- Rosson, M. B., Carroll, J. M., Seals, C. D., Lewis, T. L. (July 2002) Community Design of Community Simulations. IEEE Designing Interactive Systems DIS2002.
- Seals, C. D., Rosson, M. B., Carroll, J. M., Lewis, T. L. (March 2002). Fun Learning Stagecast Creator: An exercise in Minimalism and Collaboration. IEEE Human Centric Computing Languages and Environments HCC02.
- Lewis, T. L., Rosson, M. B., Carroll, J. M., Seals, C. D. (March 2002). A Community Learns Design: Towards a Pattern Language for Novice Visual Programmers. IEEE Human Centric Computing Languages and Environments HCC02.
- Doswell, F., Harley, H., Lewis, T., Seals, C., and Dr. G. Scales. (Feb 2002). Adapting to Life as a Graduate Student: Getting Up to Speed on Information Technology. Accepted for publication in Student Guide to Graduate and Professional School Success.
- Seals, C. and Rosson, M. B. (2001). Learning and Reuse in Visual Programming Environments: Teacher Simulation Creation Environment. Paper to be presented at HCC '01 2001 IEEE Symposium on Human-Centric Computing Languages and Environments (September 2001).
- Rosson, M. B. and Seals, C. (2001). Teachers as Simulation Programmers: Minimalist Learning and Reuse. Paper to be presented at CHI2001 (April).
- Seals, C. (2001). A Framework Supporting Educational Software Reuse: Teacher Simulation Creation Environment. Paper presented HCIC2001 (Feb).

- Harley, H., Seals, C., Rosson (1998). A formative evaluation of scenario-based tools for the learning Object-Oriented Design. *Crossroads: ACM Student Magazine*. Winter 1998, pp. 1-5.

Presentations:

- Supporting Educational Software Reuse: Teacher Simulation Builder for Teachers. Research presentation at SIGCSE 2002 Doctoral Consortium (February).
- AWC Faculty and Graduate Student Panel on Academic Careers - Panel Member - February 2002
- Presentation at Engineering Expo - Intro to Computing Careers - October 2001
- Computer Science Graduate Student Expo - Panel Member - October, 2001
- Supporting Educational Software Reuse: Teacher Simulation Creation Environment. Research presentation at SIGCSE 2001 Doctoral Consortium (February). <http://www.grinnell.edu/sigcse/sigcse2001/program.html>
- Formative Evaluation of Visual Programming Environments for Teacher Creation of Educational Simulations. Research presentation made at SIGCSE 2000 Doctoral Consortium.

Posters:

- C. Seals. (2000). Formative Evaluation of Visual Programming Environments for Teacher Creation of Educational Simulations. Poster presented at Virginia Tech Graduate Research Symposium 2000.
<http://csgrad.cs.vt.edu/~seals/research/ResearchSymposium>
<http://gsa.uusa.vt.edu/symposium/2000/abstractbook.html>
- Harley, H. and Seals, C. (1997). Object-Oriented Analysis & Design: A Novice Perspective. Poster presented at Empirical Studies of Programmers Conference.

PROFESSIONAL DEVELOPMENT ACTIVITIES:

- ACM International Student Poster Competition - Reviewer - February, 2001, 2002
- AWC (Assoc. for Women in Computing) Panel on Academic Careers - Panel Member - February, 2002 Member AWC "Association for Women in Computing" - Presentation at Girl's Day 2000, 2001, 2002
- Computer Science Graduate Student Expo - Panel Member - October, 2001
- Officer Computer Science Graduate Council 3 years, Treasurer, Social Chair, Member of Dept. Computer Resources Consortium, Member of Dept. Recognition and Awards Committee
- Secretary - Delta Sigma Theta Sorority, Inc. Blacksburg Alumnae Chapter
- Secretary - Virginia Tech BGSO "Black Graduate Student Organization"
- President, Eta Zeta Chapter Sigma Alpha Iota International Music Fraternity (Grambling State)
- Drill Sergeant, Section Leader, Grambling Tiger Marching Band (Grambling State)

HONORS

- 1998-present. Graduate Research Assistant, HCI-Visual Languages Group. Virginia Tech.
- Graduate Teaching Assistantship, Virginia Tech
- Upsilon Pi Epsilon, Computer Science Honor Society
- Software Engineering Assistantship, North Carolina Agricultural and Technical State University
- Virginia Commonwealth and Kellogg Fellowships
- Kappa Delta Pi, Education Honor Society
- Concert Mistress, Grambling Laboratory Orchestra
- Academic Merit, T.H. Harris, and Orchestra Scholarships

REFERENCES

Dr. Mary Beth Rosson, Associate Professor
Virginia Tech
Department of Computer Science
Mail Code 0106
Blacksburg, VA 24060

Dr. John Carroll, Professor and Director, Center for Human Computer
Interaction
Virginia Tech
Department of Computer Science
Mail Code 0106
Blacksburg, VA 24060

Dr. Roger Ehrich, Professor
Virginia Tech
Department of Computer Science
Mail Code 0106
Blacksburg, VA 24060

Dr. H. Rex Hartson, Professor
Virginia Tech
Department of Computer Science
Mail Code 0106
Blacksburg, VA 24060

Teaching Statement

Cheryl D. Seals

Teaching offers the opportunity to help shape the future by stimulating the minds of the workforce of tomorrow. In my pursuit of more formal education, I attended Grambling State University and received dual degrees in Computer Science and Mathematics Education. In preparation for teaching computer science, my educational foundation consists of a broad base of fundamental computing. Also I received a M.S. at North Carolina A&T State University and will shortly receive my Ph. D. from Virginia Polytechnic Institute and State University through these schools; I have been exposed to many different types of teaching strategies and styles. Drawing from this diverse experience, gives me a unique opportunity to understand why not just teaching, but “Good teaching” by teachers who care about their students is very important.

I have always wanted to make a difference in the world, and in a field such as Computer Science. I will be able to make an impact on our future society. To work competently in any field as a professional, one needs the essential tools of the trade. I feel that I have a broad educational foundation and love for the inquiry-enlightenment process. Being blessed with many God-given talents, it is my duty to be a positive role model in my community. As college professor, I would be given the means to reach many young adults, by teaching and nurturing their talent. I can strive to be a living affirmation that your dreams and goals are attainable.

I have assisted with the courses introductions to programming and intro to computer science. In assisting with these courses, I thoroughly enjoyed working with students whenever I was needed, even well after hours. I am adequately prepared to teach any of the foundational computer science classes (e.g. programming languages, operating systems). I have taught computer science and programming to high school students during my summers, which was very rewarding and challenging. As much of my research has centered on Human computer Interaction (HCI), Software Engineering (SE), with this experience I am also prepared to instruct courses in these areas (e.g. HCI, Usability Engineering, SE, and OOA&D). HCI is emerging as an important science for understanding the roles of humans and computers, and what are the best methods available for designers to exploit these relationships. Related courses of interest would be user interface design, object oriented analysis and design courses.

In addition to teaching, I believe that mentoring is very valuable for student’s success. In computer science and engineering programs there are proportionately fewer women and minorities in these areas, and I would hope to begin or to contribute to ongoing efforts of building support networks to recruit and improve the success rate of these students. During all of my computer

science studies I can only remember one woman that I had as an instructor, which at times was discouraging.

My goal as a teacher is not just to lecture, but also to inspire enthusiasm, get students to actively participate in their education, and facilitate their learning with proper educational scaffolding. For some students, computer science and mathematics courses are hard and confusing; some say, "I just don't get it!" I believe that with adequate support that almost any student that really applies them self with the proper support structure can be successful.

Research Statement

Cheryl D. Seals

My research focuses on identifying and developing strategies for reuse, which will facilitate novice programmers to create simulations. This work is realized with end user programming/visual programming techniques, and reduces the cognitive baggage of having to first learn to program to attain this goal. Currently, I am studying middle school teachers, but I want to broaden the scope of this work by making programming accessible to a wider audience (e.g. undergrad CS students, K-12 students) with End User Programming techniques.

My research is in the area of End User Programming, a subset of Human Computer Interaction that is a combination of Human Centric Computing, Visual Programming Environments, and Empirical Studies of Programmers. Empirical Studies of programmers normally studies the programming habits of expert programmers, but my approach is looking at novice programmers. The novice programmers group, which I am studying, is middle school science teachers. I am creating a framework to support their programming of educational simulations with direct manipulation and other visual programming techniques.

High-level summaries of three of my most recent research projects are given below.

Learning and Reuse in Visual Programming Environments: Simulation Builder for Teachers

(Dissertation)

End User Programming has become a popular technique to support novices in their day-to-day activities (e.g. Microsoft Word to support document creation instead of having to remember document tags in LaTeX). I am investigating how to support teachers in the Creation and Reuse of Educational simulations. There is literature on kids learning to program, but no work in the area of supporting teachers as a novice programming community. I have identified strategies for reuse in this culture as copy and past; we will build upon this and hope to enhance their productivity. In our research we have identifies specific models of reuse: Generic/Abstract and Specific/Specialized. Generic or Abstract models with basic functionality are easier to understand and appear more useful for novices and they can be generalized to more solutions than specific models. I am building a framework to validate this hypothesis.

Fun Learning Stagecast Creator: An exercise in Minimalism and Collaboration.

Seals, Rosson, Carroll, Lewis, and Colson

We are attempting to create a cross-generational learning community who will work together to design, construct, and discuss simulations of community topics. The simulations are built with Stagecast Creator, a state-of-the art visual programming

environment. As part of this larger project, we have developed minimalist training materials for middle school students. This paper reports a formative evaluation of these training materials, in which groups of students worked together on two related tutorial modules. In general the students were successful in their work with Creator, needing little aid from the experimenters, and showing evidence of enjoyment. Our aim is to develop materials that will attract participation and enable students to spend their free time and play with this environment, and as a by-product of having fun, learn more about visual programming.

Community Design of Community Simulations.

Rosson, Carroll, Seals, and Lewis

We report on a participatory design workshop in which residents of a community collaborated in learning about and designing projects for a visual simulation environment. Nine participants (five middle school teachers, four senior citizens) first conducted a participatory evaluation of a tutorial developed for the Stagecast Creator simulation tool. They then worked in pairs to brainstorm ideas for Creator simulation projects that would help raise and promote discussion of issues relevant to their community. After sharing these ideas, each pair chose 2-3 simulation ideas to refine as a specification for subsequent implementation. We discuss the participants' learning and design activities, as well as their contributions to our long-term goal of supporting cross-generational collaboration and learning through community simulation projects.