

SIMBUILDER: AN INVESTIGATION AND USABILITY STUDY OF NOVICE  
PROGRAMMING TECHNIQUES

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

---

Sumitha Kanakadoss

Certificate of Approval:

---

Juan E. Gilbert  
Assistant Professor  
Computer Science and Software  
Engineering

---

Cheryl D. Seals, Chair  
Assistant Professor  
Computer Science and Software  
Engineering

---

N. Hari Narayanan  
Associate Professor  
Computer Science and Software  
Engineering

---

Stephen L. McFarland  
Dean  
Graduate School

SIMBUILDER: AN INVESTIGATION AND USABILITY STUDY OF NOVICE  
PROGRAMMING TECHNIQUES

Sumitha Kanakadoss

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama  
December 16, 2005

SIMBUILDER: AN INVESTIGATION AND USABILITY STUDY OF NOVICE  
PROGRAMMING TECHNIQUES

Sumitha Kanakadoss

Permission is granted to Auburn University to make copies of this thesis at its discretion,  
upon request of individuals or institutions and at their expense. The author reserves all  
publication rights.

---

Signature of Author

---

Date of Graduation

## THESIS ABSTRACT

# SIMBUILDER: AN INVESTIGATION AND USABILITY STUDY OF NOVICE PROGRAMMING TECHNIQUES

Sumitha Kanakadoss

Master of Science, December 16, 2005  
(B.E., University of Madras, Madras, India, 2003)

125 Typed Pages

Directed by Cheryl D. Seals

In many existing programming languages, novices always find it difficult to transform their mental plan to terms compatible with the computer. By analyzing the ways users think to solve their daily problems and designing programming languages accordingly, would help novices to overcome this transformation barrier. This study assumes that visual programming would be more effective for novice programmers since it requires less cognitive overload than textual language-based approaches.

The study was initiated by analyzing the usability of currently available tools for visual construction of educational simulations like AgentSheets, Alice3D, 3D Game Creator, Toontalk, and Squeak. The main goal was to find the difficulties novice programmers experienced while developing simple science simulations in Squeak SimBuilder environment. These identified factors motivated and guided us to redesign

the environment to achieve effectiveness, efficiency and satisfaction of users. We followed tenets of the Natural Programming design process to accomplish this. Comparative analysis was made and a wide range of quantitative and qualitative data was collected. Results indicated that overall the redesigned environment performed better than the previous version in all aspects and showed significant improvement in fun factor, ease of understanding, and learnability.

## ACKNOWLEDGEMENTS

I would like to express my humble gratitude to Dr. Cheryl D. Seals, my advisor and committee chair, for her support, motivation, patience and insightful guidance throughout my research work and graduate study. I would like to thank my committee members for taking time out of their schedules to be on my committee. I extend my gratitude to my parents and sister for their tremendous support, love and encouragement. I am grateful to all my friends for being there and providing emotional and moral support. And, most of all, I want to thank God for blessing me with this life and guiding me throughout my life.

## TABLE OF CONTENTS

LIST OF FIGURES .....	X
LIST OF TABLES .....	XI
CHAPTER 1 .INTRODUCTION .....	1
1.1 APPROACH .....	2
1.2 DOCUMENT OVERVIEW .....	4
CHAPTER 2 .LITERATURE REVIEW .....	6
2.1 USER-CENTERED DESIGN.....	8
2.1.1 Usability .....	8
2.1.2 UI Design strategies .....	11
2.1.3 UI Design Tools .....	11
2.1.4 Learner –Centered Design for Educational Software .....	12
2.2 NATURAL PROGRAMMING.....	12
2.2.1 Why Natural Programming? .....	13
2.2.2 Learning Barriers .....	14
2.2.4 Instructional Techniques .....	15
2.2.5 Design of new language.....	17
2.2.6 LiveWorld – Environment for Novices .....	17
2.2.6.1 Message-passing protocols in programming .....	19
2.2.7 Recommendations for a programming language to be more natural .....	19
2.3 VISUAL PROGRAMMING .....	20
2.3.1 Characteristics of Visual Programming Languages .....	21
2.3.2 Comparison between Visual and textual code .....	23
2.3.3 Applications of Visual Programming .....	26
2.4 AUTHORIZING ENVIRONMENTS FOR TEACHING.....	27
2.4.1 AgentSheets .....	28
2.4.2 LEGOSheets, HyperGami and Stagecast Creator .....	30
2.4.3 3D Graphical Programming Environment .....	31
2.4.4 Squeak.....	32
2.4.4.1 Smalltalk implementation .....	33
2.4.4.2 Squeak strengths .....	34
2.4.4.3 Squeak Application.....	36
CHAPTER 3 .RESEARCH OVERVIEW .....	37
3.1 RESEARCH PURPOSE .....	37
3.1.1 Primary.....	37
3.1.2 Secondary.....	37

3.2 RESEARCH APPROACH .....	38
3.3 RESEARCH HYPOTHESIS .....	39
CHAPTER 4 .PHASE 1: USABILITY STUDY WITH SQUEAK SIMBUILDER 3.7 ..	40
4.0.1 Background study .....	40
4.0.2 Learning .....	41
4.0.3 Creation of simple science model .....	41
4.0.4 Post survey .....	42
4.0.5 Experiment Results .....	42
CHAPTER 5 .PHASE II: REDESIGNING SQUEAK SIMBUILDER 3.7 .....	45
5.1 GENERAL ENVIRONMENT .....	46
5.2 SCRIPTING .....	47
5.4 KILLER PLAYFIELD AND GROUP ERASE .....	51
CHAPTER 6 .COMPARATIVE STUDY AND EVALUATION .....	54
6.1 EXPERIMENT METHODS .....	54
6.1.1 Population .....	54
6.1.2 Apparatus and Location of Experiment .....	55
6.1.3 Experimental design.....	55
6.2 MATERIALS .....	57
6.2.1 Consent Form.....	57
6.2.2 Tutorial.....	58
6.2.2.1 Section 1: Learning.....	59
6.2.2.2 Section 2: Reuse.....	61
6.2.2.3 Interaction Guide .....	63
6.3 PROCEDURES.....	65
6.3.1 Pre-survey Questionnaire.....	66
6.3.2 Performance Data and User Observations .....	67
6.3.3 Post-survey Questionnaire .....	67
6.3.4 Retrospective Interviews.....	67
6.4 RESULTS AND ANALYSIS.....	68
6.4.1 Participant Background.....	68
6.4.2 Learning Sessions: Performance and Qualitative Data.....	70
6.4.2 Reuse Sessions: Performance and Qualitative Data .....	72
6.4.3 User Reactions .....	75
CHAPTER 7 .DISCUSSIONS AND CONCLUSIONS.....	81
7.1 Results Summary .....	81
7.2 Future Work .....	83
7.3 Conclusion .....	83



REFERENCES .....	85
APPENDICES .....	89

## LIST OF FIGURES

FIGURE 1 . RESEARCH APPROACH.....	38
FIGURE 2. OPENING SCENE IN SQUEAK .....	46
FIGURE 3. WATER CYCLE SIMULATION.....	47
FIGURE 4. OBJECT SUN AND PAINT TOOL.....	47
FIGURE 5. SUN WITH HALO .....	48
FIGURE 6 .SCRIPTING CATEGORY.....	48
FIGURE 7 . SCRIPTING WINDOW .....	49
FIGURE 8 . NAVIGATOR FLAP.....	50
FIGURE 9 . FILE FLAP .....	50
FIGURE 10 . OBJECT FLAP .....	51
FIGURE 11 . KILLER PLAYFIELD.....	52
FIGURE 12 . GROUP ERASE.....	53
FIGURE 13. OPENING SCENE IN SQUEAK .....	59
FIGURE 14. LEARNING SECTION-EXPLORING SUN .....	60
FIGURE 15 . LEARNING SECTION - CREATING VOLCANO SIMULATION.....	61
FIGURE 16 . REUSE OBJECTS IN THE OBJECTS FLAP.....	62
FIGURE 17. PHOTOSYNTHESIS SIMULATION .....	63
FIGURE 18 . INTERACTION GUIDE- A .....	64
FIGURE 19 . INTERACTION GUIDE -B .....	64
FIGURE 20. VOLCANO -MOUSE .....	71
FIGURE 21. VOLCANO - STYLUS.....	72
FIGURE 22 . STARTER WORLD TO OCEAN WORLD .....	73
FIGURE 23 . OZONE DEPLETION TO PHOTOSYNTHESIS .....	74

## LIST OF TABLES

Table 1 . Bi-polar Rating Scales .....	42
Table 2 . Likert-Scale Rating: General Ease of Use .....	43
Table 3 . Likert-Scale Rating: Assessing Motivation .....	44
Table 4 . Likert-Scale Rating: Assessing Programming Style Reaction .....	44
Table 5 . Experiment Design 1 .....	56
Table 6 . Experiment Design 2 .....	57
Table 7 . Data Collection .....	66
Table 8 . Participant Background Data .....	68
Table 9 . Learning and Creation times .....	70
Table 10 . Reuse Session Times .....	73
Table 11 . Bi-polar Rating Scales .....	75
Table 12 . Likert-Scale Rating: General Ease of Use .....	76
Table 13 . Likert-Scale Rating: Assessing Motivation .....	77
Table 14 . Likert-Scale Rating: Assessing Programming style Reactions.....	77

## CHAPTER 1 .INTRODUCTION

Usability is the first objective of natural programming. Achieving effectiveness, efficiency and satisfaction of the users for any application developed is a usable product. There are so many applications with varying complexity developed using different programming languages but the ease with which programmers code them is always questionable. Though the programming field is dominated with so many high level languages, novices always find it difficult to express their conceptual view in code. Natural programming is a technique, which helps non-programmers to easily and effectively learn a programming language. In natural programming we study culture, work environment, and the technique users adapt to solve a problem. It utilizes naturalness not only in programming but also in debugging. Success of any language greatly depends on it's learnability and productivity factor i.e. users should learn the language without any training or manual and after learning how to use it, they should be able to implement their tasks.

End-Users have been supported with direct manipulation techniques to program, where visible objects on the screen are directly manipulated with a pointing device. Visual programming is the task of specifying a program based on visual and in some cases direct manipulation techniques (graphics, drawings, animations, icons etc.). The rationale is that this removes the necessity for learning a programming language

in order to create a program or simulation. Thus visual programming tries to achieve naturalness and decreases the effort required to program by an end-user. It is to be noted that Microsoft visual programming languages are textual languages, which use a graphical GUI builder to make programming basic interfaces easier for the programmer to correct.

## 1.1 Approach

In my thesis, “SimBuilder: An Investigation and Usability Study of Novice Programming Techniques”, I studied the difficulties non-programmers undergo while developing simple science simulations in visual environments. Extensive study of the previous research done in this field of natural programming motivated me to improve the usability capabilities of the Squeak SimBuilder environment. I evaluated the usability of current state of the art systems that are used to create interactive simulations in some of the visual environments like AgentSheets, Alice3D, 3D Game Creator, SimBuilder, Squeak, and ToonTalk. Although good simulation software sources already exist in some subject areas, no interactive inquiry-based simulations are yet available for science. So some of the basic simulations in science like Volcano, chemical reactions, atomic theory etc. were created in these environments and the ease with which it can be done was studied. Pros and cons for each environment were studied.

Each of the environments is specific to certain domains. Alice3D and Squeak are deployed in the field of tutoring students and providing teaching aids. But Alice3D has its limitation that it can run only on windows platform and limited to only 2D. AgentSheets is more sophisticated environments for researchers investigating agents, robots etc. 3D

Game Creator provides different visual tools for developing games. ToonTalk for kids develop programming capability through its gaming features. Even the toughest concept is illustrated as a fun play which kids can easily grasp. SimBuilder is a tool for modeling and simulation. It is used to build and simulate science models for high school students. It is programmed in Squeak.

This research studied the Squeak SimBuilder environment in detail and utilized the results to develop easier-to-use, fun, wonderful visual programming environment designed to support the creation of educational simulations. The new environment is expected to increase user satisfaction of the system.

The Goal of this study is to redesign a system to aid teachers in delivering curricula models and also aid middle school students in their pursuit of inquiry based learning of science and/or introductory programming skills. We may utilize tenets of the natural programming design process in the study. This process includes the following:

- Identify and understand audience
- Usability study of the existing system
- Evaluate system
- Redesign based upon evaluation
- User Centered Design applied to a specific domain.

The intended audience for this study is a set of novice programmers in visual programming field. Usability study of the existing Squeak SimBuilder 3.7 version will be done to analyze the current system. As an example for the creation of an educational simulation in this environment, consider a water cycle simulation. The model of water cycle is most familiar environment in the earth science field which contains objects like:

clouds, sun, sunray, and grass. Quantitative and qualitative data collected with this preliminary study will be used to redesign the environment. A few observations were made: the menus and system flaps overload the novices with unwanted features thus confusing them; system can be made more flexible and fun to use for novices; erasing each object is cumbersome. To increase usability and to inculcate Natural programming, we redesigned the existing version to give better performance. A usability study was again conducted to evaluate the new system and the results were compared with the previous environment to check whether the proposed system is really efficient and more usable.

This research led to the analysis of opportunities and limitations of existing visual programming environment for educational simulations. Though currently, visual programming is not common compared to the textual programming this study would help students to explore this new field and hope to make their programming life easier. Future work will include porting Squeak to PDA so that it will be handy to use and can be shared easily among students in the class thus helping them in handling their projects easily.

## 1.2 Document Overview

The entire thesis is divided into seven chapters. This introduction is Chapter 1, which provides the background study and gives a high level explanation about this research. Chapter 2 is the literature review, which analyses the current literature for information relevant to this study, including summarization of natural programming, Visual Programming, Usability and educational software. Chapter 3 elaborates on the

research purpose, questions and hypothesis of the study. Chapter 4 discusses the preliminary usability study conducted with Squeak SimBuilder 3.7 version, from which the initial requirements for redesign were obtained. Chapter 5 describes the redesign and development of existing Squeak SimBuilder environment. Chapter 6 presents the methodology, experiment design, materials collected during the comparative evaluation of versions, data collection and the analysis of it. The final chapter, Chapter 7, summarizes the conclusions about this work, including recommendations for future work or enhancements.



## CHAPTER 2 .LITERATURE REVIEW

HCI as defined in text book Dix et al. (1997) is the study of people, computer technology and the ways they influence each other. We study HCI to determine how we can make this computer technology more usable by people. Research in HCI covers a wide area some of which are improving the user interface of the applications, studying the psychology of programming, Cross Cultural Interfaces, Natural Programming and Ubiquitous HCI. Brad A. Myers (1996) discussed many of the developments and major advancements in HCI which had fundamentally changed the field of computing and paved the way for future works which would help the industry to provide more usable products. My study concentrated on some of the fields of HCI, understanding the User-Centered design for any application followed by improving Natural Programming techniques for the Visual Programming environment used for educational Software.

In this ever changing world, people are required to be current with the technologies no matter their age. By reducing the strain required for learning new technology, learning can be made easy and a continuous process through out life. Reppenning (1996) identified two types of learning approach. In the constructionist approach, users have to learn themselves without any guidance. In Instructionist approach, users are provided with proper guidance to learn. Researchers wanted to make the learning more creative and interesting which is a mix of both the approaches.

However, according to Amy Bruckman and Alisa Bandlow (2002), usability is a prerequisite for learning. Before any software being created or redesigned a usability study should be done so as to make sure that the design of software is sound. Research in developing the most interactive, user friendly educational software for children which enhances their learning capabilities is one of the key areas in HCI. According to Kori Inkpen (1997), the three important characteristics to be studied are 1) The learning environment and the context of learning, 2) Usability guidelines suitable for children and 3) Gender Interaction with Computers. She found that for educational software to be efficient, the students should have fun while interacting with the software as they play computer games, and interaction techniques like drag-and-drop were not utilized as much as point-and-click with children. They found problems selecting the areas for any operation which showed that students didn't have experience using that technique. The interaction techniques girls and boys use, their approach to solve problems, and ability to solve problems differed a lot. Previous research found that if there is no proper understanding of these gender differences, design would end up creating complex environment. Practicing User-Centered design was the solution Inkpen provided for developing educational software which will understand how children of both genders interact in a learning environment. The user-centered design approach is cost-effective since in that prototypes can be developed on paper and tested before many hours and dollars have been spent for developing a product that doesn't work for a wide variety of users.

## 2.1 User-Centered Design

### 2.1.1 Usability

According to the book, “A Practical Guide to Usability Testing“ (1993), Usability means that the people who use the product can do so quickly and easily to accomplish their own task. The first step in this process is to identify the target audience and to meet with them. By conducting interviews, watching users complete tasks and listening to them talk about their work we can find out:

- what the users need
- what their work environment is like
- what is important to them
- what tasks they do both frequently and infrequently
- how they accomplish these tasks now
- how do they think about their tasks (the mental model)

Users do not have sufficient time for exploratory study and they generally expect to finish a new task easily in a given time frame. So if the application is complex the productivity will be less, ultimately users will be frustrated, the product may fail and learning curve for users will be low. Usability should be started from the first phase of development of software and the users should be involved through out the process and provide design decisions along with a usability specialist in the developing team.

Usability goals can be set early in process and at each stage the progress can be checked by testing the prototype developed. Gould and Lewis (1985) gave four principles for developing usable products:

- Focus early and continuously on users
- Integrate Consideration of all aspects of usability
- Test versions with users early and continuously
- Iterate the design

Effective design of a usable system is the most important issue. Good knowledge of design results in its success which is important for both the novices and expert designers.

Daniel Fallman (2003) argues “that the role of design in HCI must not simply be seen either as a question of problem-solving, but as an art-form, or as a bustle with reality: it is contrary to an unfolding activity which demands deep involvement from the designer.” J.W. van Aalst et al. (1995) found that a product fails in design due to some of the reasons like inadequate task analysis, weak understanding of user’s goal and inadequate set of design criteria and unsatisfactory management of the design process itself. Design plays an important role in HCI research but its role is not acknowledged fully in research. In HCI, both the academic researchers and designers from industry are involved in designing the product, so a lack of knowledge in design ultimately leads to failure of product. Design is an art of making or creating something new which didn’t exist before.

The design is captured using one of the three accounts namely the (J.W. van Aalst et al. (1995))

- process-oriented conservative account,
- product-oriented romantic account
- Down-to-earth pragmatic account.

With the conservative account, the design process builds from the requirement specification to the resulting product. The process in pragmatic account is defined for each situation. The process of romantic account is solely based on the designer's creativity and individuality. Analyzing the problem in hand then synthesizing a solution followed with the evaluation of the product outlines the stages of design. Designers thought that these rigid stages had some disadvantages where came the concept of iteration. Though iteration follows these steps it gives enough freedom to switch between the stages making design process more flexible. Contrary to the view of design as a process of certain steps, sketching reveals design as more of dialogue oriented approach. Sketching helps in transforming a designer's mental thoughts to a hard copy, which can be used to exchange views of different designers. Though sketching has its own importance in many fields, in HCI it is neglected because certain issues of sketching cannot be captured using pen and paper. To acknowledge the value of design in HCI, difference between Design-Oriented-Research and Research-Oriented-Design is important. In Research-Oriented-Design the production of new artifacts is more

important, but in Design-Oriented-Research production of knowledge is of importance. There are UI guidelines and patterns followed to produce good UI design.

### 2.1.2 UI Design strategies

Anton Elines, Gerrit C. van der Veer, and Martijin van Welie (2000) in their paper, “Patterns as Tools for User Interface Design” compares and contrasts guidelines and patterns. Guidelines are small rules which have knowledge about the design specifications and can be used when constructing new interface. But these guidelines fail in the areas of validity and applicability, at time these guidelines are unclear about when and why they are to be used. Patterns provide solutions for the problems faced by the guidelines. Patterns are proven design knowledge which is in terms of problem, context and solution. Though patterns are potentially better tools than guidelines, creating patterns for UID is difficult. Patterns for UID are similar to the patterns for software construction. They make the system more usable. The usability is the key concentration of defining a pattern. Usability in general should be effective, efficient and satisfactory to the users. Each UID pattern is formed with usability in mind. Each pattern consists of a problem, context, solution and examples. Although patterns are of interest in UID, still patterns are not widely available. Collections show writing patterns is not a trivial task though it is more effective for the design purpose.

### 2.1.3 UI Design Tools

To aid designers Landay J. and Myers B. (2001), researchers at University of California, Berkeley and Carnegie Mellon University have implemented a sketching tool

called SILK (Sketching Interfaces Like Crazy) which would help in easily sketching the interface according to usability guidelines and it allows testing prototypes so that designers can change the results when the system infers incorrectly. Usability testing found that SILK is a promising tool for early UI design. Researchers at ISTI-CNR studied that the non programmers felt comfortable designing interfaces which didn't require specifying the low-level details as in the sketch-based systems. TERESA, designed by Silivia et al. (2004) was an environment, which was useful in building and analyzing UI design at different abstraction levels and generating a suitable implementation for various platforms. It offered mixed initiative interaction together with adaptive features which avoided dealing with the low-level details.

#### 2.1.4 Learner –Centered Design for Educational Software

Amy Bruckman and Alisa Bandlow (2002) stated that user-centered design should be expanded to learner-centered design for developing educational software. This requires an understanding of not only the students who learn, but also understanding the teachers who are going to use the software. A formative evaluation which is based on usability and learning outcomes should be done to understand requirements for designing a learning environment and to guide the process of iterative design.

#### 2.2 Natural Programming

Nielsen (1994) stated Natural Programming as, “The system should speak the user's language, with words, phrases, and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a

natural and logical order”. Researchers investigated novice’s behavior to a programming language and learn how to create more natural languages to meet the user’s expectations. Natural programming is a technique which provides ordinary users with applications built the way they think. This technique helps users to get over the barrier imposed by many programming languages. Natural Programming should help users to program in the way they think in their day to day lives. Brad A. Myers in Human-Computer Interaction Institute, CMU has been doing a lot of research in developing more usable programming languages. His research is aimed at creating an environment which is more natural. A usability study was conducted by Myers (1998) to explore the state of art in the field of programming and difficulties involved. Many programming languages fail to be natural because the syntax and semantics of the language implemented never considered the words used by the people in their day-to-day life. It was found important to keep the programmers aware of what is going on with the system by providing proper responses and feedback. In his “Natural Programming” project Myers focused on studying the human side of programming and discussed design requirements of a programming language. These requirements will reduce the effort needed to write programs for novice programmers, eventually help users learn to program more easily.

### 2.2.1 Why Natural Programming?

In the fields of *Psychology of Programming and Empirical Studies of Programming*, programming is defined as “a process of transforming a mental plan that is in familiar terms into one that is compatible with the computer”. The difference between the way programmers and non-programmers think to solve a problem and the facilities



provided by the current programming languages to accomplish those tasks are great. It is pointed out by Myers (2001) that the design of new languages like JAVA and Java Script do not take into account the findings of Empirical Studies of Programmers (ESP) and Human-Computer Interaction (HCI) and follows the same mechanism for looping, conditionals, and assignments that have been shown to be the most error prone for novice programmers. Usability studies were conducted to find whether functional-oriented programming style or objects-oriented programming style best suit novice users. The Natural Programming Project aimed at studying how people naturally express the programming concepts with which guidelines for new programming language were formed. The design concepts used were from many different domains such as children creating games, teachers building educational software, office workers, military purpose, World Wide Web, etc. They found that there are gaps in knowledge about how people reason about programming, and how programming languages can be made more effective.

### 2.2.2 Learning Barriers

Myers along with Andrew J. Ko (2004) identified the six challenges inherent in developing a learnable end-user programming language. Studying at least three diverse programming systems like Alice programming environment, Visual Basic and Macromedia Flash, they identified the following barriers: design, selection, coordination, use, understanding, and information. Some of the suggestions provided by Myers et al. to overcome these barriers are inculcating creativity among the learners, a search mechanism or good documentation to find all the behaviors offered by the system,

making the learners aware of the invisible rule, designing the programming interface matching their semantics and an explanation of what the program can and can not perform.

### 2.2.3 Studying the Novice programmers

For any programming language to be more efficient, it should help novices to understand the language easily and quickly. So the way in which novice programmers think and how the implementation of environment that supports novices should be researched. Studies show that novices could understand the syntax and semantics of each individual statement but they fail when combining together those statements to solve a problem. Just understanding separate syntactical statements many not improve user understanding of concepts and technology of the new programming language. Novice programmers tend to have only a surface level knowledge of programming, while solving complex problems in a programming language generally requires a deep algorithmic understanding of the language.

### 2.2.4 Instructional Techniques

Richard E. Mayer (1981) based on his consistent results suggested a way to improve novices understanding of new technical information by providing a framework that can be used for incorporating new information and using elaboration techniques. First novices should be comfortable when interacting with the computer. In the black box approach, the computer is treated as a magic box to which input is given and output is obtained without the user knowing what is happening inside the system. In another

approach called glass box approach, the learner is aware of what is going on. To become a successful programmer it is necessary to know the details of the process going on inside the computer. A concrete model has strong effect on encoding the language which can help the learners come up with creative solutions. Elaboration techniques have been used by experimental psychologist to improve learning. The basic underlying concept is that the learners are made to summarize the new technique in their own words so that they will know how to inculcate new information with existing knowledge. Though elaboration takes more time the efficiency of programmers is increased.

James A. Spohrer and Elliot Soloway (1986) conducted programming experiments with novices to study the mistakes they generally face when trying to use small pieces of code to create a large program and the reason for their difficulties. Novices find problems learning the correct semantics because they interpret that language constructs always work in similar fashion irrespective of the application. Their difficulty in program construction arises because of many underlying problems. Some of them as stated by Soloway are Summarization problems, Optimization problems, Previous-Experience problems, Specialization problems, Natural-Language problems, Interpretation problems, Boundary problems, Unexpected cases problems, and Cognitive Overload problems. Vikki et al. (1993) studied 20 novice and expert PASCAL programmers. They found that some of the characteristics of mental representation were lacking in novices. They include a hierarchical and multi-layered approach to problem solving, explicit mappings between goal to be achieved and the code, recognition of recurring patterns, internal connection and well grounded program text.

### 2.2.5 Design of new language

The new language developed by Myers et al. (1998) at CMU will eliminate the need for most punctuation, where users normally make mistakes. This language uses a form-based technology. The languages which are easy to learn will have a direct manipulation front end other than explicit scripting. For example, in VB users can place the controls using a mouse and set their properties with dialog boxes. The new language is also self-disclosing so that when users perform an action by drag and drop the system will generate the code. These actions help the users to learn the programming language using those snippets. Using these direct manipulation features many different domains like programming for children, WWW, simulations, multimedia tools, educational software, and intelligent agents have all benefited. One study done by Andrew Jensen Ko of CMU and Bob Uttil. of Oregon State University (2003), examined that if the programming system designers offer interactive tutorial of the environment and language the initial performance of the programmer can be increased. The research on Natural Programming is an ongoing process in which J.F. Pane and Myers showed progressive findings. In 2001, they conducted studies focusing on children who were the audience for the new programming language. Children were studied to assess their learning capabilities and structure of non-programmers' solutions to programming problems.

### 2.2.6 LiveWorld – Environment for Novices

Travers et al. (1994) proposed an environment called LiveWorld, based on concrete behavioral rules and computational models that offer novice users a world of manipulable objects, with graphical objects and elements of the programs that make them

more integrated into a single framework. LiveWorld is defined as a graphical environment designed to support research in programming with active objects. The style of programming followed in this design is rule-like agents which respond to the environment. It combines object-oriented programming with a direct manipulation interface to create an environment that is concrete, reactive and flexible. This provides novices with a rich set of graphical objects, non graphical objects and elements of programming that integrate these objects. The previous agent-based languages were Agar and Playground. LiveWorld is intended to improve upon these systems without overwhelming novices with complexity. The basic structure or entity used in this interface is Frame. The design values include tangibility of objects, reactivity of objects, improving the existing objects to explore new objects and learnability. The object structure is based on frames which are a knowledge representation tool with prototype based inheritance. The advantages of this prototype based programming over traditional OOP is that it eliminates a whole set of objects and simplifies the inheritance and increases correctness.

Each frame has a name, location and set of properties. All the frames are listed in hierarchical fashion. There is no distinction between classes, slots and objects because the frame itself can act as object or a slot, thus LiveWorld permits unification unlike the traditional object system. Graphical objects are implemented by special frames called actors. The system provides a graphical library of basic actors to use. They provided both graphical as well as non graphical objects in the system which helps the novices who are

familiar with direct manipulation to make learning easier. LiveWorld also provides sensors as frames.

#### 2.2.6.1 Message-passing protocols in programming

It handles programming through a message-passing protocol over different frame objects. Action codes are written using Lisp. Messages are sent using ask primitive. The traditional top-down, command driven model of execution is not followed in LiveWorld as this top-down approach may hinder understanding of the system where control is distributed. In LiveWorld, each object has its own control and also there is a background control for the entire set of objects. But this pseudo-parallelism is not complete and needs further investigation.

#### 2.2.7 Recommendations for a programming language to be more natural (Myers et al. 1998, 2001)

- A mix of event-based and rule-based programming style to deal programming task in the same environment would be more usable.
- Instead of looping controls, aggregate operations is preferred to reduce the errors.
- Simple rules were preferred rather than complex Boolean conditional statements. It was also found that the use of negation was very less and recommended to have “unless” clause in the control structure.
- Mathematical expressions should be more natural.

- Variables were found to be more difficult and “state variables” were found to be more used to keep track of progress.
- The use of Boolean Expression “OR” and “BUT” was less frequently used and “AND” was used as sequencing word.
- A built-in list-like data structure is more natural than arrays.
- Fundamental Object Oriented capabilities are more preferred.
- Provide domain specific features in a programming language.
- Pictorial specifications should be implemented along with textual specifications during the initial stages of developing software.

The study results were used to create a new language for children. One good example of a natural programming language is HANDS (Human-Centered Advances for Novice Development of Software), which is a part of John Pane’s Ph.D. work. HANDS aids students to create simulations, games and educational software. Features like queries and aggregate operators allow many tasks to be more natural than other languages that require the use of search and iteration. User studies showed that the HANDS environment was very natural and easy to program.

### 2.3 Visual Programming

The term "visual programming" means different concepts to different people. One interpretation is that the interface or objects handled by the language are visual and

another interpretation is that the language itself is visual. In the first case, "visual language" means "language for processing visual information", or "visual information processing language". In the second case, "visual language" means "language for programming with visual expression", or "visual programming language" that improves the user interface of the programming environment and also decreases the difficulty in programming. Here we will concentrate on the second case of visual programming. Myers (1986) defines Visual Programming "as one that refers to any system that allows the user to specify a program in a two or more dimensional fashion".

### 2.3.1 Characteristics of Visual Programming Languages (Burnett et al. 1995)

- Fundamental concepts are only required to program for example pointers, variables not included.
- It has concrete programming process making everything visible to a programmer.
- Response / feedback to the programmer help in quick testing and debugging. It achieves this with help of efficient incremental translator and program execution.

The way of representing programs with logic diagram was supposed to be the first visual programming concept. Flowcharts and other graphical programming languages followed the logic diagrams. Some Visual language systems available are KidSim/Cocoa (i.e. a system designed for children), ToonTalk, AgentSheets, Squeak, Garnet, Chimera, and Forms/3 (i.e. based on the spreadsheet metaphor), etc. He also states that the visual style of programming will make students understand the concepts more clearly than textual



languages because human visual information processing is optimized for multi-dimensional data.

Whitley and BlackWell (1997) conducted surveys to show the cognitive benefits of visual programming both in academics and industry. They obtained significant results analyzing Visual Programming (VP) literature, conducting surveys with professional programmers and with LabVIEW programmers. Among many visual programming tools available, LabVIEW was chosen for a study because it is a widely used tool and previous research on it questioned some features which received the attraction of researchers. The comparison between textual and visual languages was also determined by asking a set of questions to users. VP can support a more user friendly environment and visual statements are more natural and increase the learnability of the programming language, more so than the often difficult and complex syntax offered in textual programming languages. The hope is that visual programming languages will increase the ease of learning a new programming language, and reduce time taken to develop the code to increase productivity. VPLs are primarily intended to simulate thought of human-human communication rather than human-computer interaction. Researches of VPLs should be focused not only the computational side, but also on the cognitive side. Narayanan et al. (1997) proposed a framework for analyzing the visual languages addressing the issues of comprehension, reasoning and interaction in the cognitive side and issues of visual program parsing, execution and feedback in the computational realm.

Previous research claimed that no language is universally best; rather each structure in the process of explaining the concepts obscures others. Green et al. (1992)

conducted experiments with the text based language called Nest-INE and the visual programming language LabVIEW to compare and contrast both the languages. With the results he obtained disagreed that VPLs are superior to TLs.

### 2.3.2 Comparison between Visual and textual code

The comparison between visual and textual notation of languages was studied with respect to the ease of learning, mental thought process etc. which are stated below (Myers 1986):

- Readability depends on various aspects of a language. For example, though VP is easier to read and more natural it fails when expressing arithmetic expression (less compact) where the textual languages are more preferred.
- Separate documentation like commenting is not necessary because the visual tasks are self explanatory.
- There is a vast drop in the syntax involved (few keywords, no semicolons, reduction of variable usage) thereby reducing the complexity of a language and making it easier to learn.
- The code is more modularized than textual languages. Procedural abstraction is achieved like using a form as a grouping mechanism , generalizing the sequence of operation.( Burnett et al. 1995)
- Scalability of VP is a big question. It would perform well when the project is small but tends to fail when the project size/complexity increases the same as with

textual languages. Burnett et al. (1995) showed that the solution to this problem relies not on compromising the distinctive qualities but in produce new ways to capitalize on those features.

- Human brain is optimized for visual representation more than the one-dimensional stream and VP is designed to make use of this ability.
- Visual programming increases creative thoughts and imagination helping to think outside box. Diagrams or pictures convey more than plain text.
- VPLs are based on the flowchart notation with which many people are familiar with and so transforming their ideas existing in the physical world to the computer domain make it easier.
- Studies found that the inherent programming concepts were not visual but the pictures helped with the understanding of the abstract principles.
- VPLs can be used for teaching the programming concepts for individuals with who have problems with reading comprehension and for the physically handicapped.
- Green et al. (1992) concluded from his studies that the graphics involved in the VP consumes more time than process a textual code.
- Dynamic type checking is achieved in most of the VPLs like the TLs but there are obstacles for implicit type checking.

- Achieves Data persistency in four different ways to extend the lifetime of data beyond a single program.
- Numerical ratings were analyzed for usability aspects and the computational aspects of textual and visual representation. It was found that VPLs were rated highest in their ease of writing and the power of coding. The results also showed that for the users who were aware of VPLs knew their importance, advantages.

Some of the disadvantages of Visual programming (Myers 1986)

- Visual Program representation requires more space and memory than the textual representation
- VPLs take longer to execute and also occupy more space.
- Unstructured programming practices are allowed
- Static representation of programs is hard to understand and editing is difficult. Burnett et al. (1995) proposed a preliminary solution to overcome this problem by devising static and dynamic syntax to be similar.
- No provision for comments. Burnett et al. (1995) says that textual documentation like one line comment is possible but it has a problem of screen real estate in which there is limited visual interaction.
- Even though visual data abstraction is achieved partly, more support is needed for user-defined visual appearance and interactive behavior. (Burnett et al 1995).

- Event handling is slow in VPLs.

Naraynan et al. (1997) in his study on visual programming in Human-Interaction perspective says that “there is no need to choose between visual and textual languages as they are two extremes of spectrum spanning from text to illustrated text, to annotated pictures and to purely visual representation.”

### 2.3.3 Applications of Visual Programming

Taking advantage of VPLs in various domains, they are used in a number of applications like building educational software, image processing, signal processing etc. (Jurgen Herczeg, Hubertus Hohl and Matthias Ressel 1993). Visual programming technique is the most natural and appropriate for building graphical user interfaces. Tools like interface builders are used for creating GUIs by means of direct manipulation. These interface builders act as a visual front end to a special programming language for creating, manipulating objects and they are not visual programming tools. Some recent tools used faced some problems like:

- The dynamic aspects of UI design not supported.
- Tools are suitable for specific applications, they are not generalized.
- There is no scope for redesigning or modifying the already existing interfaces.
- Customizing interfaces at run time is not supported.

The new approach given by Jurgen (1993) provides powerful and knowledgeable UI toolkits which provide a rich programming interface. These tools can be used to remodel interfaces which have been already created by the conventional programming. This concept of remodel interfaces is built in user interface development environment XIT, based on CLOS (common Lisp Object System) and the X windows System. This system includes user Interface toolkit that provides an object-oriented programming interface. It consists of both low-level and high-level frameworks. User Interface browser is for inspecting the structural dependencies of UI and its underlying applications. Researchers in France, Olivier et al. (1995) developed a visual programming tool called WHIZZ'ED editor which helps in creating highly interactive objects to build interfaces. Whizz'Ed is actually built on the concept of data flow diagrams and with a set of predefined elementary components.

#### 2.4 Authoring Environments for Teaching

On reviewing the literature and research currently going in the areas of Natural Programming, Novice Programming and Visual Languages, it became necessary to study the usability of various educational tools available in the market, which were developed to support novice learning and using visual language techniques. With the limited number of experienced teachers in some field authoring environments may help students to learn properly and efficiently. These tools are like having effective tutors. Some of the systems studied are Squeak, Alice 3D, Lego Sheets, Agent sheets and Stage cast Creator. Jan Erik Moström (2002) analyzed the use of concurrent constructs in authoring environments for teaching, stated few features needed for a successful authoring environment:

- Support for reuse.
- Revision control to handle changes.
- Support for collaborative work.
- Different tools for help during authoring, for example support for patterns.
- Flexibility to allow last minute changes.
- An easy to use search facility.

#### 2.4.1 AgentSheets

AgentSheets is widely used by students as educational software and also by graduate students to research about life long learning. Agents are instructed by the end users explicitly to do certain actions. The users who do not have strong programming background can make use of these agents to develop tools. Using Domain-oriented Languages and Domain-oriented agents facilitate the working of agents. Alexander Repenning and Andri Loannidou (2004) developed AgentSheets environment which is based on the spread sheet metaphor is used for designing DODEs (Domain Oriented Design environments). HyperGami is also one such language. Actually AgentSheets is not a visual programming environment but it helps in developing visual programming systems. AgentSheets combines several programming techniques into what it calls “Tactile Programming”. Repenning and Ambach (1996) describes “*Tactile Programming*” as: *Tactile Programming extends the framework of visual programming by adding perception by manipulation.* This means that statement, values, variables, etc., of the programming language are objects that are manipulated by dragging and dropping them to their desired place. While building interfaces using conventional programming

techniques, a new interface element cannot be obtained from the existing ones but AgentSheets overcomes this problem. Each element in the grid structure is called an agent. Agents in AgentSheets have the capability of multimodal communication. They respond to speech, keyboard input and webpage content. Every agent consists of a sensor which has methods to be acted upon; Effectors which are a communicator, state, depiction and an instance. A rule-based language called visual Agent Talk is used in AgentSheets to communicate with agents. In one example, "The boulder Mountain biking Advisor" the agents respond to a voice input and outputs the suitable places for biking in Boulder County. In addition the agents call methods to check the weather condition suitable for biking and tell whether biking is suitable in that area. In "bridge Builder" example, the agents are used to demonstrate basic understanding of the static and dynamic forces on the bridge. Users can explicitly instruct agents through rule based language to accomplish certain applications.

AgentSheets was also studied in a real world application for its usefulness and its naturalness (Repenning 1991). KEN is an expert system, which is used to configure power plants. Configuration charts are used by the experts to query the knowledge base. A particular scenario was given to a set of users and they compared the AgentSheets-based Chart representation to the conventional text-based representation. They noted the following: increase in performance, accomplish complex tasks, supports easily the mapping between function required in visual form, and provides inheritance. AgentSheets proved to support complex applications in the field of expert systems and also provided an incremental approach to building agents from existing ones.



#### 2.4.2 LEGOSheets, HyperGami and Stagecast Creator

LEGOsheets, a rule-based programming environment implemented in AgentSheets, is developed by the MIT media lab for children to create mechanical artifacts. Gindling et al. (1995) discovered that children were encouraged by a lively application with colorful icons and audio for either positive or negative feedback. LEGOSheets provides an introduction to programming and designing of mechanical artifacts like motors, vehicles, robots in conjunction with sensors and effectors to program the behaviors of those artifacts. LEGOSheets tends to help children by helping them to sharpening of their basic skills. But it fails in the cases of programming where sequencing and timing of events are important.

Hypergami is another design-oriented language in the domain of math integrating with LEGOSheets discussed by Repenning et al. (1996). Both the direct manipulation and scheme programming is employed in HyperGami. It supports a wide range of mathematical forms from simple geometry to 3D shapes for modeling. It helps people to creatively approach learning math and problem solving. Since both these environments stimulate learning capabilities, improvements to network these environments should be implemented.

Stagecast Creator formerly known as Cocoa and KidSim can be classified as a graphical-rewrite-rule language. It is based on the movie metaphor, where users create a cast of characters who interact and move within a simulation micro world. Seals et al. (2002) studied the usability of Stagecast seeking to improve the programming skills of non-programmers. They found that the language fails in the areas of reusing rules, and

students had problem with matching the visual state of rule to the exactness of visual syntax.

#### 2.4.3 3D Graphical Programming Environment

Mathew Conway et al. (1999) discussed a 3D Graphical Programming environment. Alice 3D was designed to overcome the difficulties faced by the previous environments and to give a new experience of learning with more ease and find more compelling examples to aid students in learning to program. It is primarily a scripting and prototyping environment that allows the users to build virtual worlds and write sample programs to animate objects in those worlds. 3D interactive simulation can be accomplished through the following steps:

- Opening a new world,
- Populating the world with the objects and focusing the camera,
- Defining the functions for each object through sets of commands,
- Creating scenes with more complicated functions.

An Alice animation begins with an opening scene, created by populating a virtual world with objects, which have six degrees of freedom/orientation. World can be selected from some of the provided environments. Once the opening scene is set up, the next step is to plan and write a program for animating interactions between the objects and each other and between the objects and the virtual world in which they reside. Alice defines an assortment of built-in actions. In general, actions can be subdivided into two categories:

- Those that tell an object to perform a motion.
- Those that change the physical nature of an object.

All commands in Alice are animated by default whenever it is semantically reasonable. A program/script is a list of instructions for the objects to perform an action. Many usability studies were conducted with the Alice software and the scholars and identified many of its more interesting features. Cognitive load is reduced for extremely common operations through the removal of X, Y, Z from the API, replacing these terms with the more useful and more Lego-like direction names and surface names of Forward/Back, Left/Right, and Up/Down. Implicit threads in Alice make launching parallel actions easy. The resize operation and the space scaling operation are both useful, but are independent and orthogonal, even if using a 4x4 matrix in the implementation makes this separation difficult to build.

#### 2.4.4 Squeak

Squeak is not just a programming language, an Integrated Development Environment, and a meaningful authoring environment for kids over 5 years of age. “Squeak is a movement towards an environment where you have separate areas to explore – the large number of behaviors, 3D graphics, musical synthesizers – and yet all in a uniform, general, powerful framework” said Mark Guzdial, an assistant professor at Georgia Tech’s College of Computing. Squeak, an open source environment was developed in 1996 by a team at APPLE for the need of Smalltalk language to pursue many applications providing a proper user interface which can be programmed even by

non-programmers, non-technical people and children. NASA center for distance learning used Squeak to implement interactive Web activities for NASA's KSNN and NASA CONNECT programs. Guzdial and Rose (2002) described the learning philosophy in Squeak, which promotes learning for school kids. First, the users experiencing the fun of learning by creating or viewing multiple representations of the same processes helps in understanding different conceptual levels. Secondly, Squeak supports dynamic processes implemented with multimedia and allows new approaches to learning by supporting the increase of creativity of the students.

#### 2.4.4.1 Smalltalk implementation

Ingalls, Kaehler, Maloney, Wallace and Alan Kay worked together to develop Squeak. Squeak is the first Smalltalk system that is completely self-supporting compared to other commercial Smalltalk implementations like Apple SmallTalk, SmallTalk/V etc. Every Squeak release comes with an image file, virtual machine and complete source code. To achieve this useful level of performance, the virtual machine is written in Smalltalk and a translator is used to convert to C, leading to an interpreter in Smalltalk itself that can dynamically increase processing speed. Smalltalk is the preferred environment for research and development because of its rich class library and sophisticated programming tools which makes it an attractive environment for rapid prototyping, for experimenting with interactive applications (Mary Beth Rosson 1990). Squeak is portable, malleable, full-service computing environment, including browsing, split-second recompilation and source debugging tools, all in a 1-MB footprint. It is able to support the intimate computing potential of PDAs and the Internet.

#### 2.4.4.2 Squeak strengths

Some of the features of Squeak which add strength to the environment are:

- Efficiency and scalability of for large projects, and supporting all object formats of the exciting Smalltalk system, a need of compact and general object memory was required. A full 32-bit object pointer to every object was implemented to achieve the desired effect.
- A two-generation approach followed in Apple Smalltalk was applied to get good garbage collection behavior, there are a number of challenges like its capability to deal with variable length headers, remapping of objects pointers as a method for achieving incremental garbage collection that leads Squeak to be usable for real-time applications like music and animation. Squeak's garbage collection took only 250 milliseconds, which is very small compared to system of similar and larger sizes.
- Squeak's BitBit supports a wide range of color depths namely 1-, 2-, 4- and 8-bit table-based color, as well as 16 and 32 bit direct RGB color. It also supported anti-aliased image rotation and scaling.
- In real time to achieve sound and music synthesis, sound generation methods were written to run directly in Smalltalk.
- To achieve interactive graphics WarpBit is completely described in Smalltalk, and then translated into C to deliver suitable performance.

- Entire VM is approximately around 100 pages. Squeak performance was obviously seen with respect to byte code/sec and sends/sec which was partly due to removal of scaffolding such as assertion checks and range checks on memory references – and partly to improve the running model of the translator.

Dan Shafer (1996) discussed that despite many advantages and performance increases, programmers accustomed to C, C+, and Java would find the Squeak Smalltalk syntax a little cumbersome. Squeak, more than trying to attract the Java, C++ programmers it aims to increase the support for the first time users. Four aspects considered for improvement by the Squeak central were:

- Experiments with alternative syntaxes
- Making it easier to create simple applications
- A new, streamlined programming framework focusing on an integrated :object operating table”
- Integration of “SqueakToy” scripting tiles.

#### 2.4.4.3 Squeak Application

Squeak is used in the following fields (Ned Konz, 2004):

- Education – 80000 users in Spain and tens of thousands of users in Japan and around the world.
- Web applications – XML support, file system, networking capabilities.
- Academic research- Squeak's portability, share ability and malleability caught the attention of many in academic research
- Multimedia
- Croquet – Squeak is used as the basis for Croquet system.

## CHAPTER 3 .RESEARCH OVERVIEW

### 3.1 Research Purpose

#### 3.1.1 Primary

- Determine the key factors with which novices found difficulty using the Squeak Environment (Squeak SimBuilder 3.7).
- Design a new user interface for Squeak based on the results obtained from the previous usability studies.
- Conduct a usability study with redesigned interface.
- Compare the data obtained with the previous study and analyze whether the new design increased the user satisfaction, ease of learning, and fun.

#### 3.1.2 Secondary

The secondary purpose was to examine whether multimodal input i.e. using mouse and the stylus would increase the performance of users, making it more fun to learn and investigate.



### 3.2 Research Approach

This research can be subdivided into three tasks as illustrated in Figure 1. In the First Phase, a pilot study was conducted for analyzing the usability of Squeak SimBuilder 3.7 Version. The result of this analysis was used to redesign the environment in the Second Phase. In Second Phase, a new design was implemented. Once the environment was redesigned, in the Third Phase experiments were conducted and the results were compared with the previous pilot study. The three phases were designed to answer the research hypothesis as discussed in the following section.

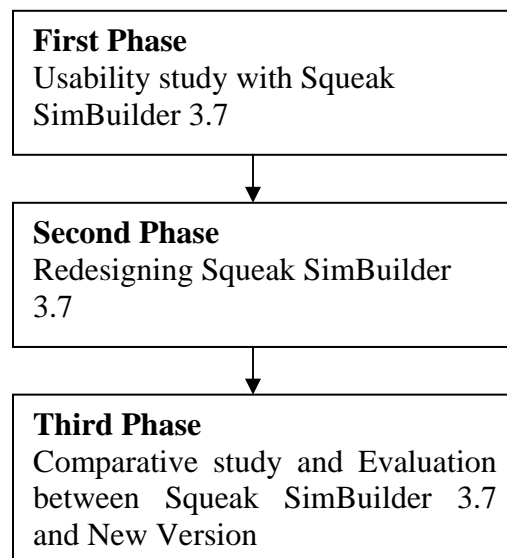


Figure 1 . Research Approach

### 3.3 Research Hypothesis

#### Hypothesis 1:

There will be significant increase in the ease of understanding compared to existing Squeak SimBuilder 3.7 Version.

#### Hypothesis 2:

There will be an increase in tool associations and the new interface will make it easier to remember the location of tools.

#### Hypothesis 3:

There will be a significant increase in the flexibility of the environment with the use of stylus than the mouse.

#### Hypothesis 4:

The new interface will have increase user rating of their fun during building simulations.

## CHAPTER 4 .PHASE 1: USABILITY STUDY WITH SQUEAK SIMBUILDER 3.7

The preliminary stage of this research was to explore the Squeak SimBuilder environment and to conduct a usability study on the existing system from which the new design requirements were obtained.

### 4.0.1 Background study

The pilot study was held in spring 2005. It started with a pre-survey to understand the participant's background and their computer literacy. There were a set of 9 students from computer science majors with good programming background in Java, C, and C++. Most of the participants were well versed with personal computers with an average of 13 years of computer experience. They had previous experience in using some drawing software and also with visual programming. Some of their ideas of the role of computer in classroom were:

- It can play a vital role if implemented properly but only risk is it can do more instructing than the instructors.
- More for illustration purposes than for analysis.
- Speed up experience and control for the order and quality of those experiences.
- More important in industrial engineering where simulation can reduce the cost.

Participants felt some of the diagrams from various sources like geology textbooks; photographs, weather, plan design, working schedule, pilot training etc can be simulated.

Compared to the participants of the first study, students in the second study were equally proficient in computers and they were comfortable using it.

#### 4.0.2 Learning

Participants were made aware of the objective of this study and their role in this experiment. Students started their learning session with the help of a 5 page minimalist tutorial, which gave the basic ways to create objects, add behaviors and the ways to interact with other objects. The tutorial described each step with the pictures showing what would be the result so that it will help the users in expecting the solution. It also had an interaction guideline, which gave helpful hints to remind them of frequently used tasks on what they have learned. The participants were asked to ‘think aloud’ during the session so as to understand their thoughts.

The learning tutorial began with exploring the water cycle model and the participants were asked to modify the behavior of the objects in it to get good understanding of how to create scripts. During user’s learning, they were required to speak out using the think aloud protocol to express their thoughts and what they expected as outcome. Their questions were recorded to analyze the user’s thought process.

#### 4.0.3 Creation of simple science model

Following the learning session, users were given a real world model of a volcano eruption. They were asked to create a simulation for it using the tools provided. With the basic functionalities learned, the users were tested for their understanding and the system was evaluated to see whether it can be suitable for the novice programmers.

#### 4.0.4 Post survey

After the completion of volcano simulation, a post survey was conducted to know the user's reaction towards the software to analyze the positive and negative things about the environment, which would help us in redesigning the environment to make it more efficient and user friendly.

During the whole experiments, quantitative and qualitative data were collected, such as time, errors and critical incidents, user's comments for the purpose of analyzing. Each session took approximately took one hour for each participant to complete the entire study.

#### 4.0.5 Experiment Results

Users provided their satisfaction with the environment by answering set of question, which used two types of rating scales. The first set of five questions asked the participants to provide a rating on a bi-polar scale to analyze the usability of the environment. The means and standard deviations for these rating are shown in Table 1.

Table 1 . Bi-polar Rating Scales

Bi-polar Scale	User Ratings Mean (SD) N=10
Terrible ----- Wonderful	3.5 (0.71)
Frustrating ----- Satisfying	3.3 (0.67)
Dull ----- Stimulating	3.4 (0.84)
Difficult ----- Easy	3.4 (0.85)
Rigid ----- Flexible	3.4 (0.84)
Boring ----- Fun	3.8 (0.92)

A quick review of the means show that the users overall rated the environment to be fun a wonderful experience, with lower marks for satisfying experiences. A high fun factor shows positive indication of the user friendliness of environment for kids.

The second set of questions used the Likert-scale to obtain user’s reactions specific to the learning activities. Participants responded using a 5-point scale from 1=Strongly Disagree to 5=Strongly Agree. Most of the items were written with a positive context such that a “5” would be a positive reaction; some items written with a negative context are noted using italicized text, and the ratings for these items have been recorded to be consistent with the others. Table 2 assess the general ease of use , which reveals that the system provides good understanding but not that easy to get started and familiarize with the simulations. Table 3 assesses the motivation provided by the environment for users. All Table 3 items have good mean rating supporting the environment. The means in the Table 4 reveals that the tools made it harder to convert their thoughts to simulations.

Table 2 . Likert-Scale Rating: General Ease of Use

Likert-Scale Rating	User Ratings Mean (SD) N=10
Easy to learn and use	3.3 (0.67)
Easy to get started	3.2 (0.92)
<i>Hard to remember tool location</i> (NOT)	3.0 (0.94)
Easy for novices	3.3 (0.95)
I understand how to use	3.7 (0.48)
<i>It was hard to recover from errors</i> (NOT)	3.1 (1.29)

Table 3 . Likert-Scale Rating: Assessing Motivation

Likert-Scale Rating	User Ratings Mean (SD) N=10
Fun for building simulations	3.4 (1.07)
Creation of working simulation	3.7 (0.95)
I can have objects any size I want	3.7 (0.82)
I am enthusiastic about creating simulation	3.7 (0.82)

Table 4 . Likert-Scale Rating: Assessing Programming Style Reaction

Likert-Scale Rating	User Ratings Mean (SD) N=10
<i>Drag and drop rules were complicated (NOT)</i>	2.9 (0.99)
Simulation works logically but tools made it harder	2.7 (0.82)
Rule creation was simple and natural	3.7 (0.67)

## CHAPTER 5 .PHASE II: REDESIGNING SQUEAK SIMBUILDER 3.7

Analyzing the results from the usability study conducted with Squeak SimBuilder 3.7 environment, we found certain features, which need re-designing that, will help the users to perform better. The main aim was to improve the usability of the environment for the novice users. Squeak is an open source project and can be freely downloaded from [www.squeak.org](http://www.squeak.org). It supports many operating systems like: Windows, Macintosh, UNIX and etc. Squeak SimBuilder environment has undergone many developments in the past few years. The environment is a combination of the following files: “image” file, text source code, text change file, and an executable virtual machine file. The environment can be updated either by filing in and compiling a set of changes or by distributing a new version of the image and/or source code. The base version used for redesigning was Squeak SimBuilder 3.7. The changes made were filed in to develop a new interface. The flaps, the killer play field, and the group erase were the features redesigned in our new environment. The opening scene in the environment is shown in the Figure 2.



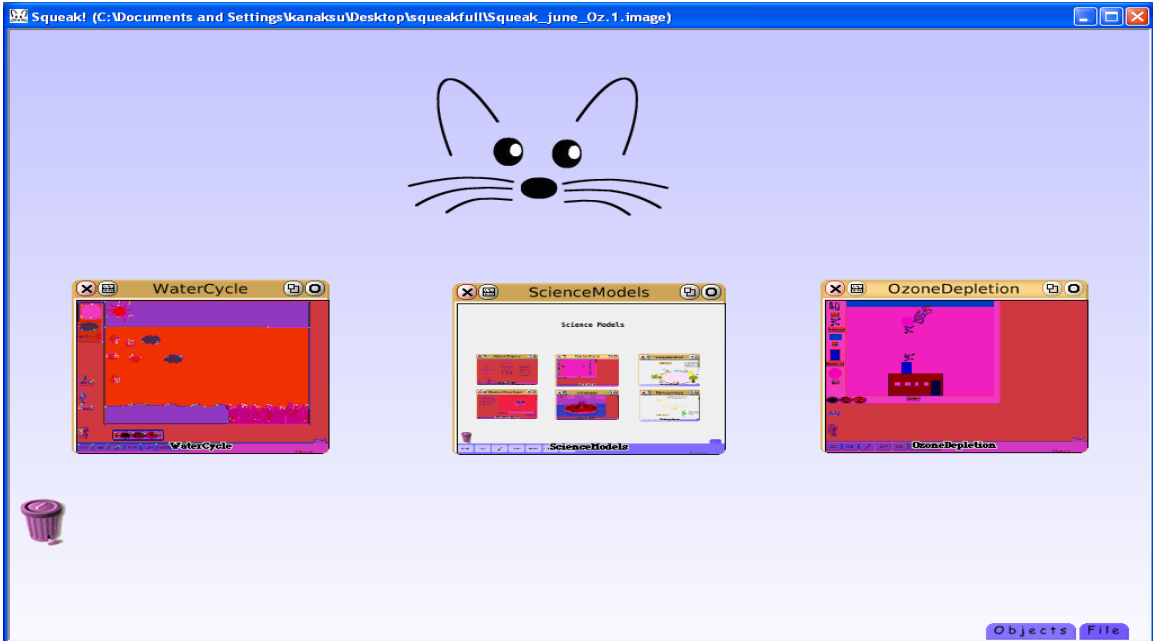


Figure 2. Opening scene in Squeak

### 5.1 General Environment

One of the main problems faced by the novices during the usability study was that too many windows open up and user may feel they lose control over the environment. The user's get frustrated, which reduces their motivation to continue using the software. To avoid this confusion, the new interface is well organized as shown in Figure 2. The WaterCycle and OzoneDepletion are separate simulations and the Science Models contains six other simulations within it. Figure 3 shows the WaterCycle model. To run the simulation, users hit "go" and see that the Sun produces Sun rays, the clouds move forward producing rain and the mist coming up from the sea. The Science model contains simulations explaining the chemical reaction, atomic theory, etc.

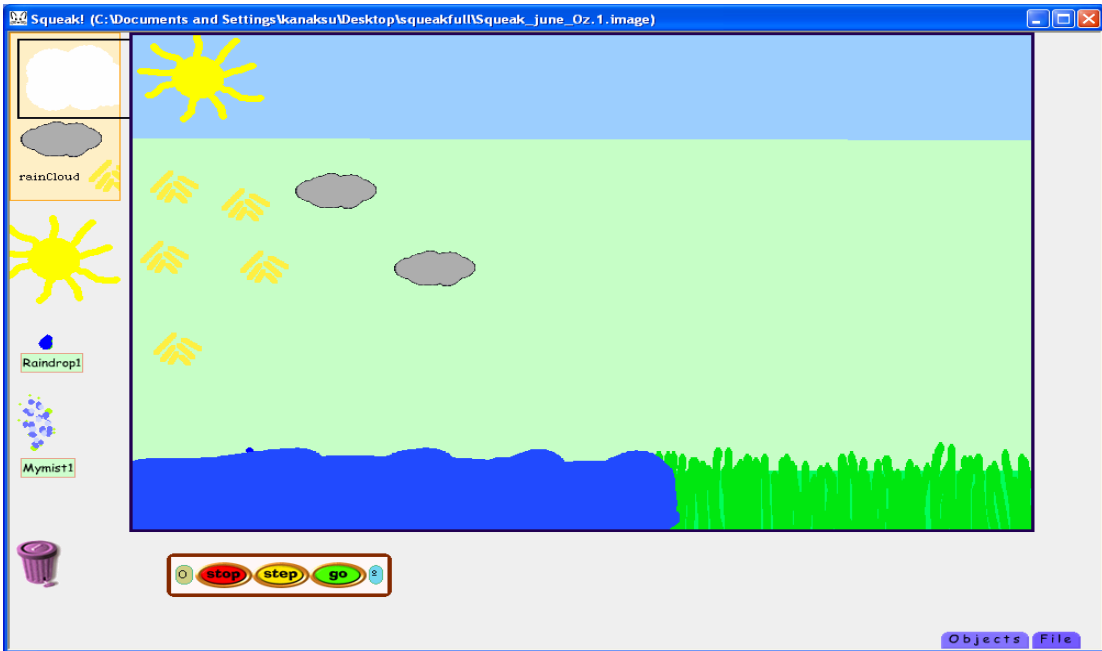


Figure 3. Water Cycle simulation

## 5.2 Scripting

A project in the Squeak SimBuilder environment starts with the user designing a visual representation of a simulation they plan to create. There can be many objects that will interact to create a working simulation. For example they can draw object (Sun) with the help of paint kit provided (Figure 4).



Figure 4. Object Sun and Paint tool

After creating the object Sun, its behavior can be added using the scripting tool. The user invokes the halo around the object using <Alt> and <Middle button> click (Windows OS) and click the “blue eye” to open viewer as shown in Figure 5.



Figure 5. Sun with Halo

Figure 6 shows the viewer for the object and the categories of scripts. This reveals all the scripting categories and the rules available for this object. There are thirteen categories of scripts available for adding behaviors. The user programs by dragging rules from this viewer to the rule window. To make a script active, the user selects the rule, drags it and drops it to the world.

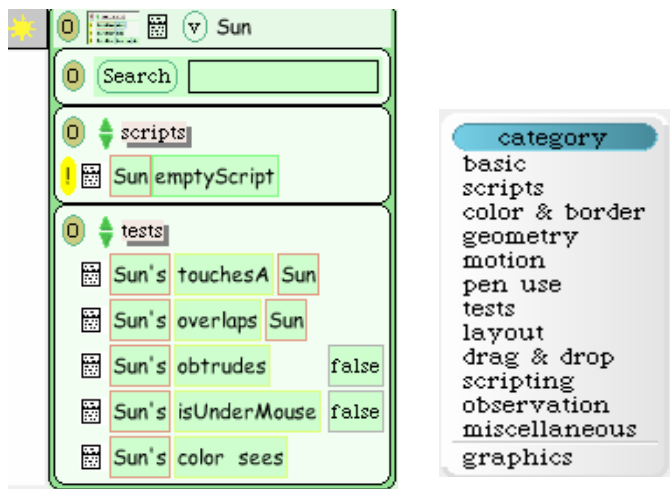


Figure 6 .Scripting Category

To make the Sun move forward, drag the “forward by” from the basic category and drop it in the world and change the “normal” to “paused” as shown in Figure 7. Users can test their scripts once with “!” for incremental execution. In similar fashion, rules can be added to all the objects in a simulation and can be played when all the scripts will start running.

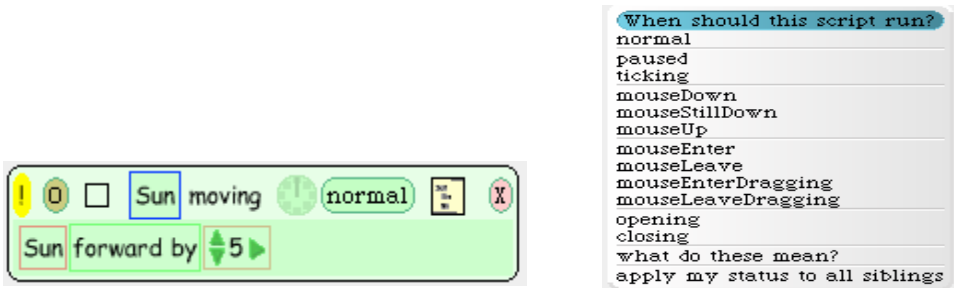


Figure 7 . Scripting Window

### 5.3 Flaps

The current version of the Squeak SimBuilder environment as shown in Figure 2 opens with the Squeak icon in the top with three other projects (WaterCycle World, Science Models, and Ozone depletion). In Squeak SimBuilder 3.7, the interface has five flaps namely: Squeak, Navigator, Widgets, Supplies, and Tools. The Squeak flap contains many options (e.g. save , about and trash can. The Tools flap contains objects useful for a developer. The Supplies flap contains forms, buttons, e-book and etc. The Widget flap contains objects useful for multimedia application. The Navigator flap contains few basic buttons for creating projects. Since the goal of this study was to find how usable and effective this environment is for the novices, and from the results obtained from usability study with Squeak SimBuilder 3.7 the new design is aimed not to over load users with unnecessary features or options which are not required while getting introduced to the

new environment. The Tools, Supplies and the Widgets flaps are hidden and the trash from the Squeak flap is placed in the opening scene. The items in the Navigator flap (Figure 8) are used to redesign a flap called File and added one more global flap called “Objects”.

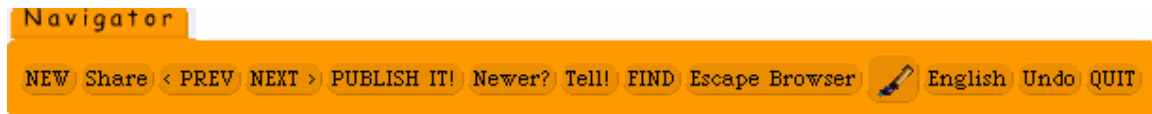


Figure 8 . Navigator Flap

Some of the features in this Navigator flap are not required for the novices, which we got rid of and created a new global flap called “File” similar to the look of Windows menus and it has only the required options useful for novices as shown in Figure 9.



Figure 9 . File Flap

The File flap contains only seven basic functions without overloading the new programmers.

- OPEN - To find existing projects in the system.
- NEW – To create a new project.
- PAINT BRUSH – To make painting.
- <PREV – To go to the previous project.
- NEXT> - To go to the next project.
- PUBLISH IT! – To save a project.
- QUIT – Exit squeak.

The File menu is programmatically added to the environment by evaluating the ProjectNavigationMorph and addFileFlapIfMissing. Separate methods for each of the actions in the File menu are reused from the already existing methods. The methods were modified as required. Another global flap called “Objects” is also added to the environment which contains the basic tools required for running simulation and the objects which can be reused. Figure 10 shows the Object flap. It contains the script runner for running simulations and other objects like Sun, Sky, Chemicals, BrCl, Smokestack, and Chemicals which are reuse objects from one of the existing simulation called “Factory Model”. It aids the novice user, by providing an easy mechanism to reuse generic objects rather users starting objects completely from scratch.

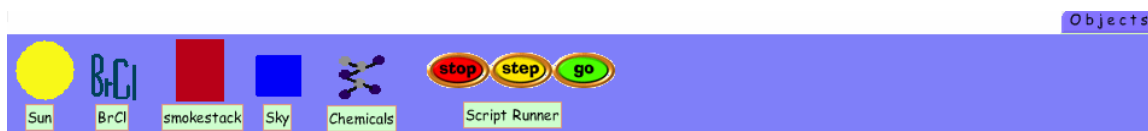


Figure 10 . Object Flap

#### 5.4 Killer Playfield and Group Erase

In Squeak SimBuilder 3.7, we wanted the objects to be deleted when they hit the boundary, a script should be added to that particular object to check for that condition. The users can either make the object wrap when it hits the boundary or delete itself. Instead of writing scripts for deleting the objects, if the playfield could handle this procedure itself, it will decrease the burden for the novices. One problem with adding the delete script explicitly to the objects is that even when the object is removed from the environment and placed in the trash the process is not actually removed; it still runs and degrades the performance. Figure 11 shows the killer playfield which deletes the object

which hits its boundary automatically. By using this killer playfield, the performance of the Squeak can be increased by not wasting process speed for objects in the trash.

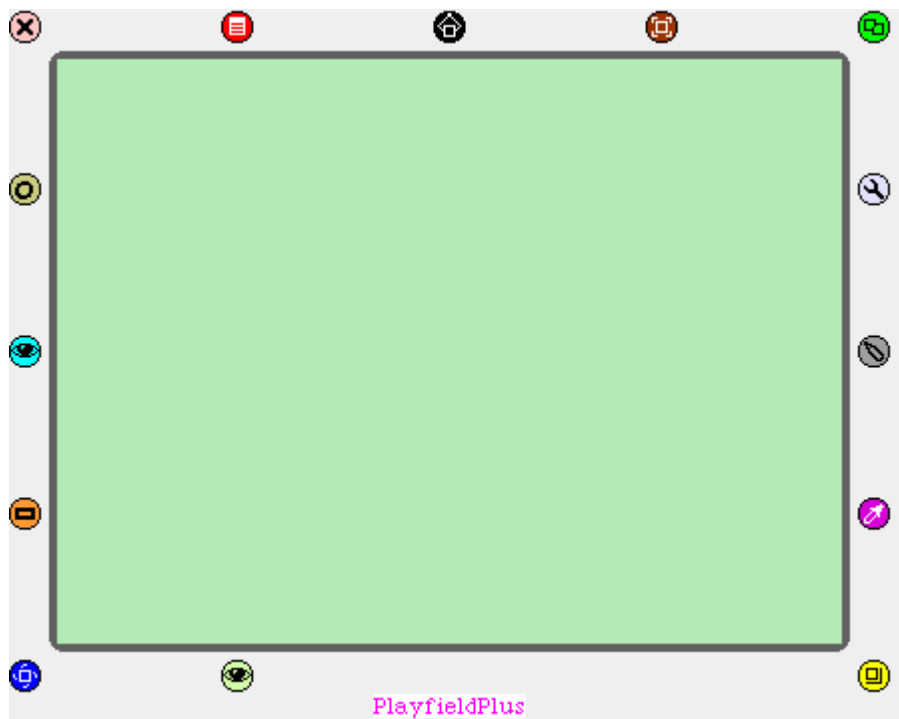


Figure 11 . Killer Playfield

Erasing each object individually by bringing the halo around the object and hitting (X) was time consuming and users were frustrated when there were many objects to be deleted. So there came the need of group erase, which could delete all the objects at a single time. Group erase as shown in Figure 12 is accomplished by holding Shift along with the Left Mouse button click (Windows) and dragging across the objects to be deleted. A new halo appears showing a blue box like boundary selecting all the objects. Once the selection is done hit the (X) to move all the objects to trash.

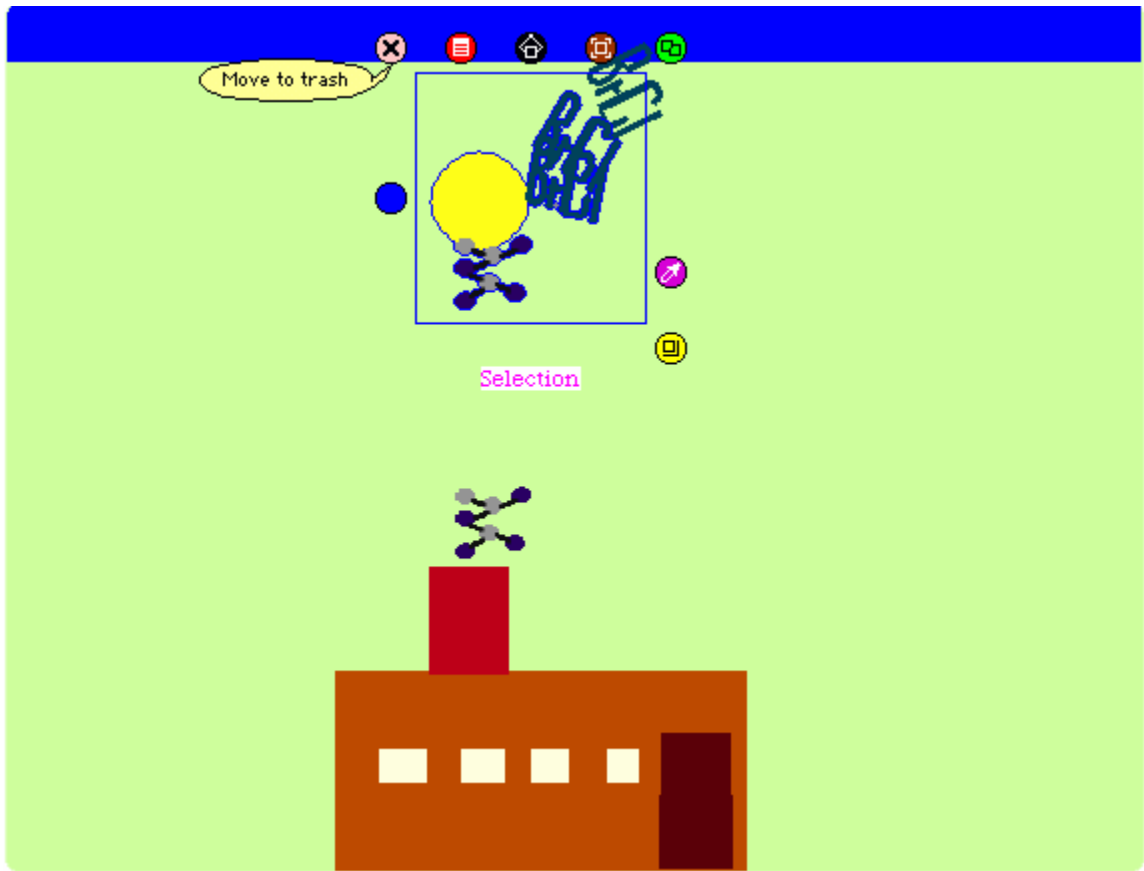


Figure 12 . Group Erase

The main difference between the Squeak SimBuilder 3.7 version and this new environment is focused on usability and support of the environment. Most of the changes are aimed at the novice programmers, and not overloading them with unwanted tools or information with the technique of keeping advanced tools hidden from initial user view.



## CHAPTER 6 .COMPARATIVE STUDY AND EVALUATION

Some of the valuable findings from the usability study conducted in spring 05 helped in redesigning the Squeak SimBuilder environment to make it more efficient for the novices. A similar usability study was conducted with the new environment and the results were compared with the previous study. The major goal was to determine if the new design and interface of Squeak SimBuilder would produce more satisfaction, ease of use, and fun for the novices. The secondary goal was to test whether multimodal input (mouse and stylus) would increase the usability of the system. The following section describes the methodology followed when conducting the experiments, the instrument used to capture the required data, analysis of the results and the comparison with the previous study using statistical tools. The results of comparison would conclude whether the new design satisfies the hypotheses of this study.

### 6.1 Experiment Methods

This section discusses the methodology followed in this experiment. The study of population, apparatus used and the design will be detailed here.

#### 6.1.1 Population

The intended population for this study is set of 12 undergraduate or graduate students above the age of 19 years enrolled in Department of Computer Science and Software

Engineering, Auburn University. Each participant considered for this study will have good background knowledge about the computers and should have used computers for more than 10 years. They are considered novice programmers in this study because they would not have had experience using visual programming tools. An announcement was made in the class and the volunteering students for this study take part in the experiment. This experiment will run for approximately 1.5 hours. Though no monetary compensation will be provided for the participants, they will be exposed to new field of programming through an educational tool.

#### 6.1.2 Apparatus and Location of Experiment

All of the study sessions were conducted at an Auburn University's Computer Science Department. The office utilized was located in the Old Power Plant (OPP 108). The study was conducted on an IBM machine with 17'' Monitor running Windows XP equipped with standard scroll mouse and an inbuilt speaker. A stylus (I-pen) with a stylus pad was provided for half of the participants. An Evaluator monologue was pasted on the wall for the participants to read before starting the experiment. Two evaluators were normally present during the study to observe participant reactions while exploring the environment.

#### 6.1.3 Experimental design

The experiment was a comparative study of Squeak 3.7 version and our new prototype design. The entire experiment was divided into two phases, the learning phase and reuse phase as described in Table 5.

Table 5 . Experiment Design 1

Squeak SimBuilder 3.7 Version	Learning	30-45 minutes
	Reuse	30 minutes
New Version	Learning	30-45 minutes
	Reuse	30 minutes

In the learning phase participants were introduced to simulation building and basic functionalities of Squeak SimBuilder. They explored the Water cycle model with guidance of a tutorial to help with their understanding of how to write scripts to achieve desired action. They were asked to create interactive simulations and utilizing the think aloud protocol. . An interaction guide was also provided to easily identify the tools required to create new simulations. At the end of learning session, the participants were asked to create a basic Volcano simulation using the experience obtained by exploring Water cycle method. During this creation time they were not allowed to ask any clarifications from the investigator. The Factory model and Science models were also presented if the participant wanted to explore more simulations in the Squeak environment. After a short break, participants were asked to take the Reuse session. The Ozone model was studied and the objects in it were reused in creating the Photosynthesis World Simulation and users also created original objects when necessary. We followed a lattice structure to design the entire experiment. 12 participants (P1 to P12) were divided into two groups with respect to the interaction style as shown in Table 6.

Table 6 . Experiment Design 2

Interaction style	Reuse Method	
	Ozone->Photosynthesis	Starter -> Ocean
Mouse	P1	P10
	P2	P11
	P3	P12
Stylus	P7	P4
	P8	P5
	P9	P6

## 6.2 Materials

This section details the materials required for the study. An informed consent form and tutorial were given to the participants.

### 6.2.1 Consent Form

The Institutional Review Board (IRB) of Auburn University requires that any research involving the human being, surveys etc. should be approved for its validity as research. After the approval from IRB, an informed consent was given to the participants when they attended the experiment. It lists out the purpose, objective of the study, the motive behind running this experiment, risks involved and the compensation for same and the benefits. The consent form is signed by all the investigators and in case of necessity; the participants can contact the investigators in future through email or phone.

The participants sign the form and can take with them after completing their study. (See Appendix A for details of Informed Consent).

### 6.2.2 Tutorial

After participants signed their consent form, they were provided with a 15 page tutorial. The main purpose of the tutorial was to introduce SimBuilder environment to the participants and to help them to create small projects. It clearly pointed that there was no need of any visual programming experience for working in this environment. The entire tutorial was drafted as a power point presentation with 22 slides. Each slide is a guided exploration card taking a user through one programming concept. The starting page discussed the purpose of study and includes a screen shot of opening environment in Squeak SimBuilder as shown in Figure 10. The first section explored some of the existing simulations and behaviors of the objects. The second section explained the reuse of objects between different projects. The tutorial aims to decrease the learning effort for understanding anew programming language. Each step in the tutorial was discussed with a screen shot of what actually happens when they play around the environment. By this visually the participants captured more information than with lengthy texts. Sufficient help was provided at each step.

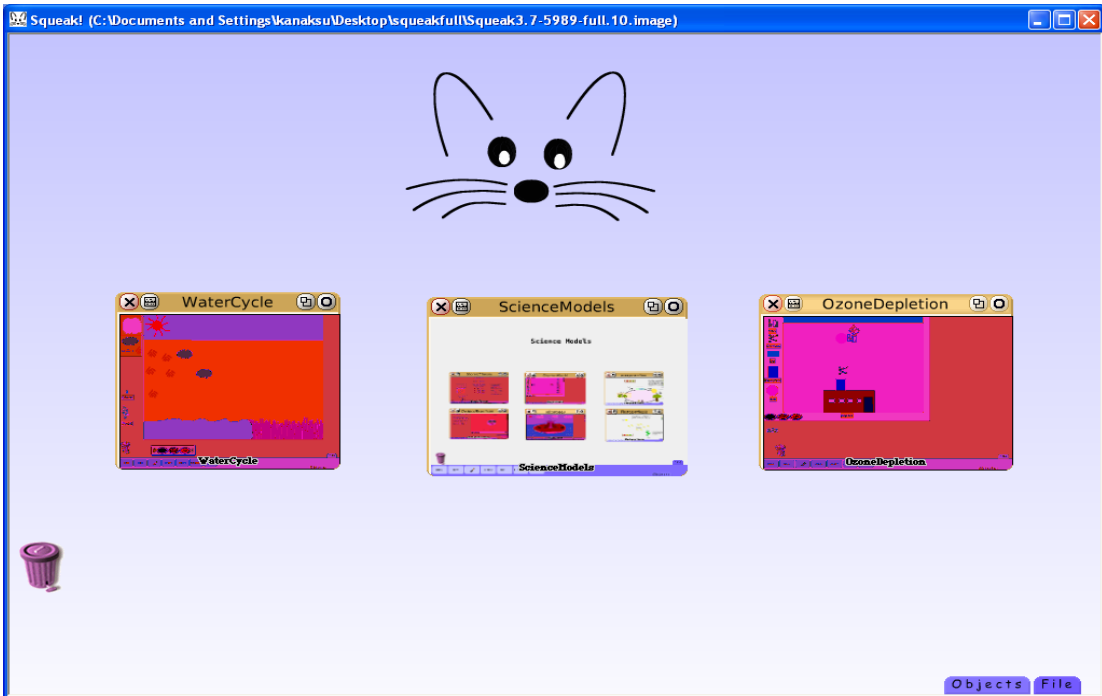


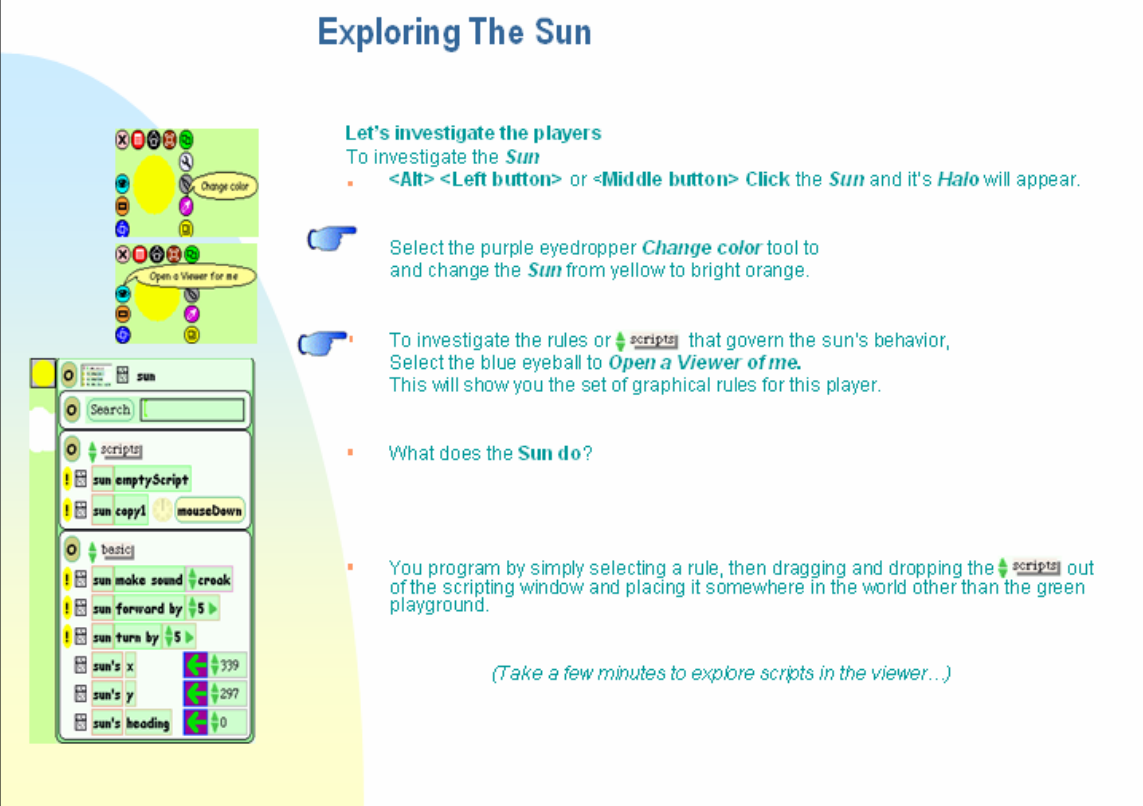
Figure 13. Opening Scene in Squeak

#### 6.2.2.1 Section 1: Learning

This section consisted of 10 slides. It started with exploring the WaterCycle model which was one of the simulations placed in the opening scene of Squeak SimBuilder. Participants were asked to run the simulation and observe how different objects like sun, cloud, and rain interacted with each other. After they explored the complete simulation, they were asked to observe the behavior of each object in it. The scripts for Sun (Figure 11) and other objects were expanded as training examples. The user was instructed in how to activate objects by invoking the objects halo to manipulate it. The objects halo provides the user with the following operations: open viewer, change color, rotates object, etc. Creation of new scripts by direct manipulation techniques (i.e. dragging and dropping from the viewer) was also discussed. With the guided exploration, participants were asked to change the direction of the cloud and also to increase the speed

of its movement.

### Exploring The Sun



**Let's investigate the players**  
To investigate the *Sun*

- **<Alt> <Left button>** or **<Middle button>** Click the *Sun* and its *Halo* will appear.

Select the purple eyedropper **Change color** tool to and change the *Sun* from yellow to bright orange.

To investigate the rules or **scripts** that govern the sun's behavior, Select the blue eyeball to **Open a Viewer of me**. This will show you the set of graphical rules for this player.

- What does the **Sun** do?
- You program by simply selecting a rule, then dragging and dropping the **scripts** out of the scripting window and placing it somewhere in the world other than the green playground.

*(Take a few minutes to explore scripts in the viewer...)*

Figure 14. Learning Section-Exploring Sun

After the learner explored the WaterCycle simulation, he or she was asked to draw new objects like a bird to the playfield using the paint brush tool available in the file flap and create behaviors or methods to make it fly in the simulation. Participants were encouraged to think aloud and ask for help to clarify their doubt until this period. A performance task was designed to assess the experience obtained by exploring the WaterCycle model. Participants were asked to create a sample environment, which simulates the volcano eruption. Figure 12 shows the tutorial page, which describes the steps for starting a new project and getting the playfield, script runner for the environment. After 5 minutes of brain storming and drawing a paper prototype, they

proceeded to create a simulation. No clues or guidance, or help was provided during this process. The degree to which they succeed in developing this environment measures the effectiveness of tutorial and grasp of concepts in short time span.

**Creating a Volcano Simulation**

- To Leave WaterCycle  
Select File and Press <PREV.
- Now that you are back in the Welcome page.  
Select File and Press NEW project
- Click Unnamed1 at the bottom of the new window and Replace it with volcano\_yourinitials.
- Click to begin a New Project.
- Drag the Playfield Plus and Script1 from the Objects
- 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.

Figure 15 . Learning Section - Creating Volcano Simulation

#### 6.2.2.2 Section 2: Reuse

This section consisted of 7 slides. The task for this section was to use the objects provided for creating two new simulations called Ocean World and Photosynthesis. A set of objects were provided in the Objects flap which can be reused. Our goal was to identify how helpful those objects if reused and make suggestions of other objects, which could be added to make it more generic and more widely reusable. In the first step, participants were asked to investigate each player in the Object flap to discover its behavior. Figure 13 shows the reuse objects in the flap. The next step was to create an









Ocean World simulation using existing object templates and also adding new objects and behavior.

## Reusing the Objects

There are set of objects provided in the **Objects** Flap which can be reused for other simulations.

- Investigate each player to discover its' behavior.



-  The Mover just moves in one direction.
-  The Emitter produces another agent.
-  The Eraser erases other agents that it contacts.
-  The Replacer replaces the Mover with another agent.
-  The Changer will change another player into a new player when it comes in contact with it.


-  Refer to interactions guide for *Help*.



Figure 16 . Reuse Objects in the Objects flap



After the Ocean world, the Ozone Depletion model was investigated. In this model, a Factory object creates pollution, which moves upward in the air until it comes into contact with Sun. When the pollution object overlaps a chemical reaction takes place and converts to elements Bromine and chloride, which upon contacting the Ozone layer will deplete it. Some of the objects in this simulation model like Sun and Factory can be reused for Photosynthesis model. Figure 14 shows the steps in creating the Photosynthesis model (i.e. the factory is a producer or emitter of other objects which can be generalized and reused for other models).


## Creating new agents 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 reusing existing players and adding new agents. We will begin with the small task of reusing a new *player* from the Objects flap.

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

**Task 2.**  **new players**  Refer to interactions guide for *Help*.

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

**Think of other interactions to make your Ocean World simulation interesting.**


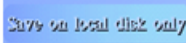

- Press  and then 
- Press  and return to the Welcome page.

Figure 17. Photosynthesis Simulation

### 6.2.2.3 Interaction Guide

To help users to remember significant features required for creating simulations, a two page guide was provided. It contains a pictorial representation of icons with their meaning, buttons, dialogue boxes and menus. This guide will aid participants by providing support for them to perform the task easier and more quickly. Figure 15 and 16 shows the Interaction Guide.

## Interaction Guide (Object Halo & Handles & Paint Tools)

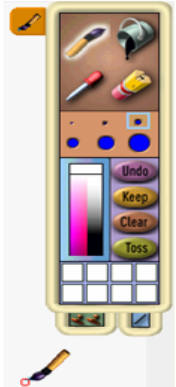
### Halo Tools

To manipulate objects in Squeak *SimBuilder* select the object and **<Alt>< Left button>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.



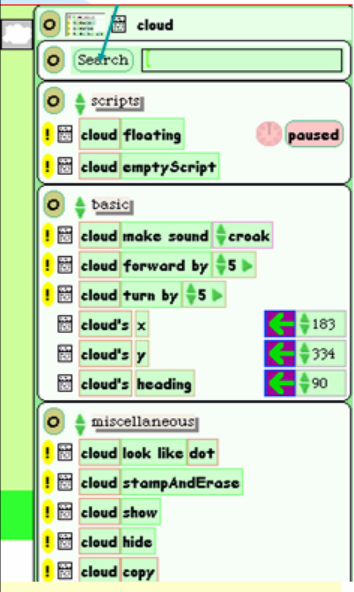
- Use the **Paint Brush** to create.
- Use **Paint Bucket** to fill areas.
- Use **Dropper** to select a color.
- Use **Eraser** to modify.
- **Multiple Circles** choose brush size.
- **Color palette will change color.**
- Press **Keep** when complete.

Figure 18 . Interaction Guide- a

## Interaction Guide (Object Behavior & Scripting Tools)

### Scripting Tools

To open more tools click




### Saving your Work

**To Save your projects.**

- Press **PUBLISH IT!** and
- then **Save on local disk only**

### Scripts define behaviors



A user defines the behavior of objects by creating a script for that object.

To use one of the predefined scripts. **Select a script** and **drag** it from the script window into the world any place other than the playground.

Press **!** to **Test** an individual script.

### Scripting Categories

The **Search** area allows you to quickly locate a script with the search.

The **scripts** category is where user created scripts are located.

The **basic** category is to make sounds and move your object.

The **color & border** category is to make sounds and move your object.

The **tests** category contains scripts that help you to test conditions.

The **graphics** category is to make sounds and move your object.

The **miscellaneous** category contains many scripts (copy, show, hide, delete, etc.)

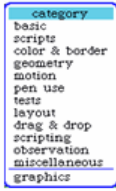


Figure 19 . Interaction Guide -b

### 6.3 Procedures

A general announcement was made in the undergraduate and graduate classes in the Department of Computer Science and Software Engineering at Auburn University about this study for requisites of participation. We aimed for a population of about 12 students for the study. A time slot was allotted for each participant so that there was little waste of time and they can reschedule their daily work accordingly. The experiment started with the participants reading the consent form, which described the purpose, objective of this study, and informed them in what needed to be accomplished at the end of this study. After reading the informed consent, students decided whether to participate or not and if they wished to participate they signed the form and kept it as reference for future in case they wanted to contact the investigators. Next, users took an on-line pre-survey, which was used to understand their background details like major, experience with software etc. After the completion of pre-survey, the tutorial was provided and the investigator gave a formal introduction of the environment, they were insisted to think aloud when browsing exploring the environment.

The experiment started with a learning session, which approximately ran for 30 to 45 minutes. The tasks for this session were to familiarize the user with the environment by guiding their investigation of rules and behaviors of the WaterCycle Simulation and to demonstrate user's understanding by creation of a Volcano Simulation. During the creation of the Volcano Simulation, no clues or help were provided. After learning session, a 5-10 minutes break was provided for the participants to refresh. Next in the reuse session, which was designed for 30 minutes, users were asked to create two

simulations – Ocean model and Photosynthesis model which reused objects from the Objects flap and from the Ozone Depletion model.

After completing the experiment, participants were asked to take a post-test questionnaire, which gathered subjective reactions to the environment measuring the satisfaction they obtained using this environment. A retrospective interview was conducted finally to allow the investigator to collect any last participant thoughts. The following table (Table 7) shows the overview of the data collected through out the experiment.

Table 7. Data Collection

Method	Description
Pre-survey Questionnaire	User Background, Major, Computer Literacy.
Performance Data	Time taken, number of rules created, error recovery.
User Observations	Qualitative Observations
Post-survey Questionnaire	User reactions and system ratings
Retrospective Interview	Understanding Users thoughts

### 6.3.1 Pre-survey Questionnaire

The main purpose of this survey was to collect background information about the participants. For some questions, participants answered by entering text, some had yes or no options and others used Likert scales. Basic details like Gender, Age, Major, etc. were collected initially. Their educational background was in the Computer Science field, all had experience in computer programming, and years of experience using computers was also asked. A Set of questions analyzed their familiarity with particular software like

drawing software, word processors, spreadsheet programs, chat etc. A set of questions were designed to assess their perceptions about simulations, use in the real world ,what could be best simulated and the role of simulating software in teaching. Pre-survey Questionnaire took approximately 10 minutes to complete. (See Appendix B for details of Pre-survey Questionnaire).

### 6.3.2 Performance Data and User Observations

Performance was analyzed through out the learning and the reuse session. Time taken for creating the new simulations, the number of rules added, the complexity of rules, questions asked by the participants while learning, number of objects reused during the reuse session were some of the data collected for analysis purposes.

### 6.3.3 Post-survey Questionnaire

This was designed to gather user's reaction towards the environment and to assess performance of the software. Overall user reactions to the system were obtained using six bi-polar rating scales. The six scales were: Terrible/Wonderful, Frustrating/Satisfying, Dull/Simulating, Difficult/Easy, Rigid/Flexible, and Boring/Interesting. Other questions used Likert Scales to rate the ease of use, fun creating simulations, easy to get started, and was it easy for the novices There were some questions asked about the tutorial we also planned to use this information to redesign it to better help future users.

### 6.3.4 Retrospective Interviews

It had set of questions that obtained the last thoughts about the environment, any suggestion, pitfalls, and good things about environment, which they wanted to share.

## 6.4 Results and Analysis

This section provides the results, both quantitative and qualitative data and analysis. We start with the summary of participant's background obtained from the pre-survey and then discuss the results obtained in both learning and reuse section. We also compare the data sets with the previous usability study with Squeak SimBuilder 3.7 and provide discussions for the same. In the following discussion "N" specifies the number of participants involved.

### 6.4.1 Participant Background

A set of 12 students was recruited from the Computer Science Department to participate in this study. Their pre-survey provides information about their background. A summary of several quantitative measures appears in Table 8.

Table 8 . Participant Background Data

	Quantitative Measures N=12
Average age	24
Percent Female	67 %
Average years of computer use	11

The ages of 12 participants ranged from 21 to 28 with a mean age of 24 years. All had good experience in Software design and Computer programming like C, C++, and Java. Half of the participants had previous teaching experience by teaching courses in the school. In answer to the question regarding their comfort using computers, majority of the participants rated them as "good with computers".

All of the participants had considerable experience with computers, with a mean of 11 years and mostly with the PCs and much less with MACs. The majority of users have used computer games and the drawing software (e.g. Adobe Photoshop, Microsoft Paint, AutoCAD, etc.). They saw a role for computer simulation in the classroom with the responses like the following:

- Can be used for handicap students
- Teaching children, more hands on, more visual than text books
- Simulation of rain
- Simulation portraying the generation of computers
- A good instance would be for physics/chemistry classes. Often these classes present issues or topics that a student may not be able to physically see or observe. A simulation would allow the students to visualize the concept presented.
- Computer simulations could play an excellent role that will give a more visual outlook of environmental issues and how things actually work, taking children far beyond their imaginations.

When asked what kind of simulations they would build, the participants proposed the following: volcano, butterfly lifecycle , photosynthesis, earth quake, tornados, caterpillar to a butterfly , driving a car - a simulation of this would be very helpful and much safer when teaching someone to drive , earthquakes, thunderstorms, hurricanes/tornados, melting ice-caps, smoke filled lungs , network of computers , tsunamis, tornadoes, earthquakes, hurricanes , manufacturing unit in a factory, working of a computer network, modeling of social organizations



#### 6.4.2 Learning Sessions: Performance and Qualitative Data

The learning phase started with a guided exploration of the Water cycle method followed by the users creating a volcano simulation. Table 9 shows the times (in minutes) for the participants completing the learning and creation phase using Squeak SimBuilder 3.7 and the new version. Learning time was measured from the time the participants began reading the tutorial until they completed the creation of bird and adding behaviors to it. The creation time started when they started reading about the simulation until they published their project.

Table 9 . Learning and Creation times

Squeak	Squeak SimBuilder 3.7	New Version
	Average Minutes N=9	Average Minutes N=12
Learning (Water cycle)	17	16
Creation (Volcano)	18	14.5
Total Learning time	17.5	15.25

Comparing the means of both studies, the new version shows a decrease in the total learning time. Though the time taken for exploring the Water cycle was mostly the same in both the studies, the creation time showed a significant decrease. This shows that the new version has provided a well guided tutorial during the exploring phase.

Analyzing the volcano simulation created by the participants we observe that an average of 5 objects were created and simple rules like forwarding, making sound was used.

During the exploration of the Water cycle model, qualitative observations were made. It was noted that few participants had problems with dragging the scripts from the viewer pane to the playfield and insertion of a particular script under the empty script. The learners had little trouble in activation the object halo for a specific object.

The drawing tool was helpful to all the participants. Half the participants used the stylus for drawing and the rest, used mouse. Participants were more comfortable using the mouse rather than the stylus because it was more sensitive and they don't have prior experience in using the stylus. Figure 17 shows the volcano simulation created using mouse.

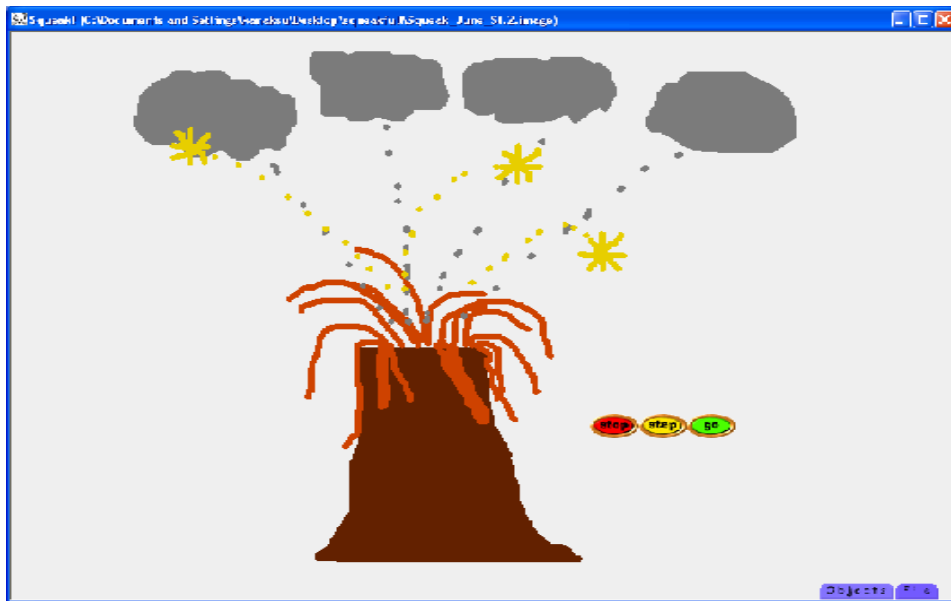


Figure 20. Volcano -Mouse

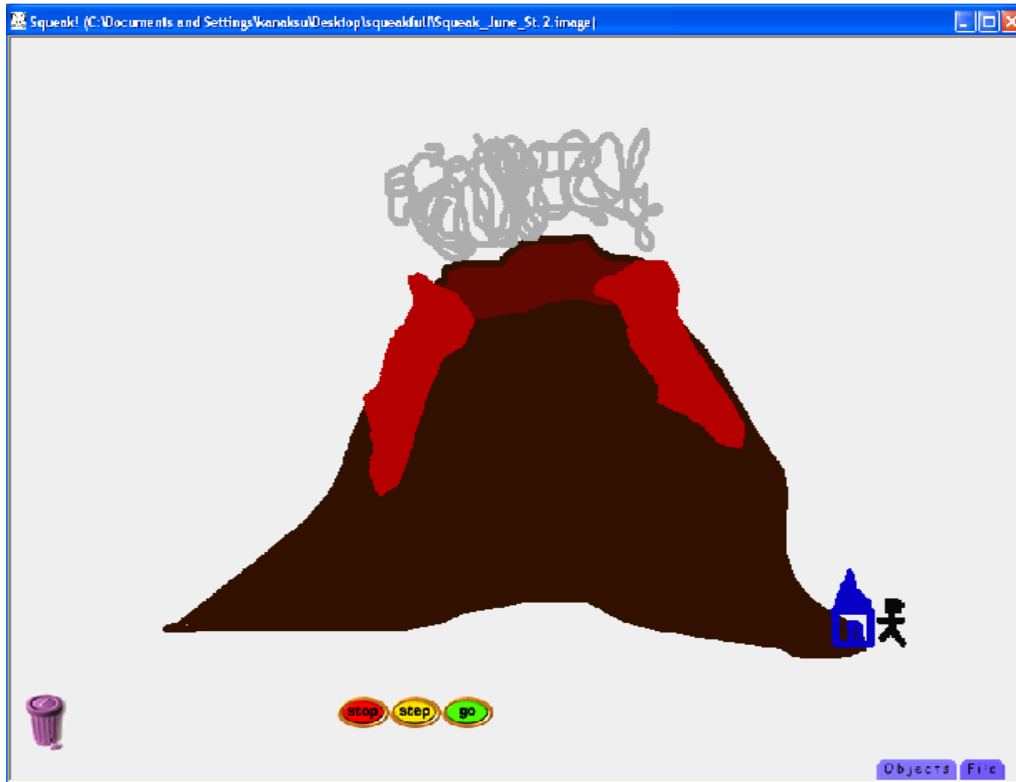


Figure 21. Volcano - Stylus

Figure 18 shows the volcano simulation created using stylus. During the learning session, we observed that learners were benefited using the paint brush, erase tool and the undo option. Users were confused about drawing the objects, which needs to interact separately, and keeping it. They also had little problem in viewing the behaviors of objects they created.

#### 6.4.2 Reuse Sessions: Performance and Qualitative Data

The reuse session was conducted in two different ways. Half the participants (six) explored the Starter world simulation and reused the world to create the Ocean world simulation. Other half of the participants explored the Ozone Depletion simulation and

reused the objects in it to create the Photosynthesis model. Table 10 shows the average time taken during the reuse session either ways using the new version of Squeak.

Table 10. Reuse Session Times

Reuse Method	Average Minutes N=12
Starter world to Ocean world	17.3
Ozone Depletion to Photosynthesis	16.3

The mean difference in times, between the two methods is less showing that both the base simulation (Starter world and Ozone Depletion) were comparatively efficient for reuse. In the first reuse method (Starter world to Ocean) as shown in Figures 19 the users reused the objects emitter as ocean, mover as wave and replacer as sand. Though the user did not complete the entire simulation of reusing all objects, they found that just by re-designing a few of the base objects it would be easier to create new simulation. Similarly, Figures 20 shows how the objects in Ozone Depletion simulation can be reused for the Photosynthesis. Users reused smokestack as sun, chemicals as rays, and sun as plants. With the few changes in the scripting behind the base simulation, they made the new simulation work.

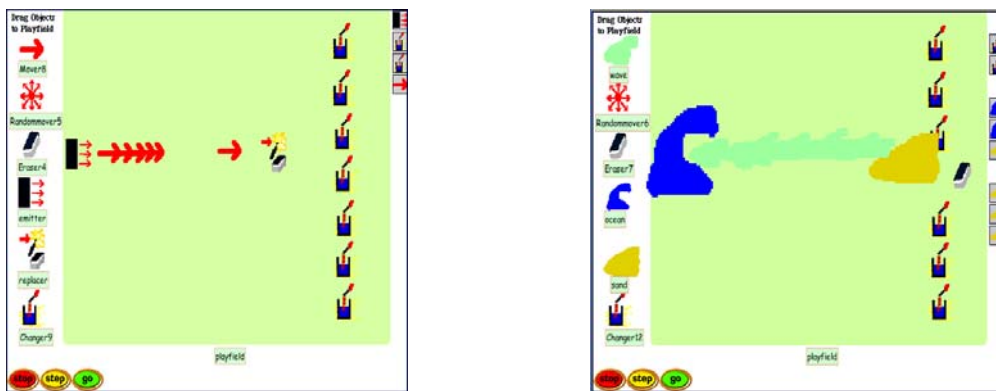


Figure 22 . Starter World to Ocean World

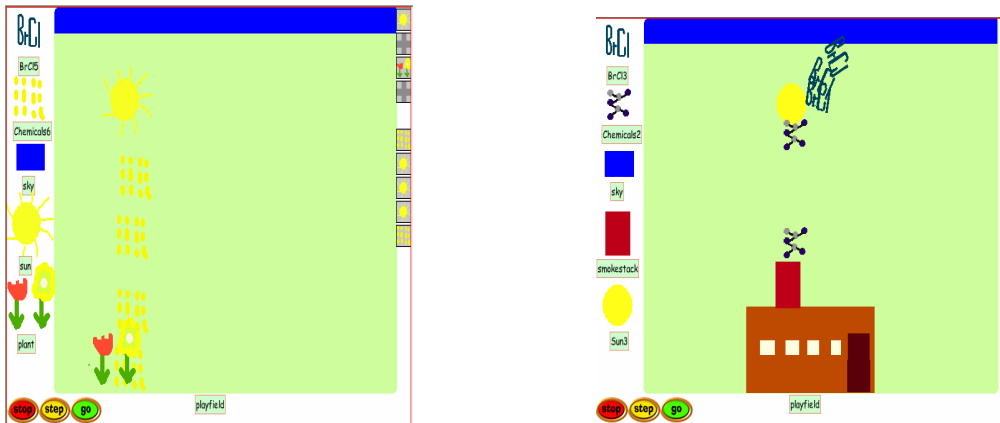


Figure 23 . Ozone Depletion to Photosynthesis

During the reuse sessions, qualitative observations were also made. The participants started their first task by exploring the model simulation provided (Starter world or Ozone depletion) followed by drawing their target simulation (Ocean World or Photosynthesis) in paper. They were directed to assess if and how the objects in the model simulation could be good candidates for reuse in creating their new simulations. They began with re-drawing the existing objects and changing the supporting scripts to suit their new simulations. Users found that reusing existing objects was very helpful and easy. We found that participants understood the semantic concept behind each object easily by using the emitter (emission of something) to Sun (emitting rays) or Ocean (emitting waves). In contrast, the visual representation of smokestack in the Ozone Depletion model did not visually mean the emission function.

Comparing both the reuse models provided, we analyzed that visual representation of objects in the Starter world is more convincing than those in the Ozone depletion model. Users preferred the objects emitter, mover in Starter world than objects smoke stack, chemicals in Ozone depletion because those visually implied their function.

### 6.4.3 User Reactions

Users provided their satisfaction with the environment by answering to a set of questions, which used two types of rating scales. The first set of five questions asked the participants to provide a rating on a bi-polar scale to analyze the usability of the environment. Table 11 shows the mean and standard deviation for these ratings, broken down by the version of Squeak used.

Table 11. Bi-polar Rating Scales

Bi-polar Scale	New Version Mean (SD) N=12	Old Version Mean (SD) N=10
Terrible ----- Wonderful*	4.0 (0.74)	3.5 (0.71)
Frustrating ----- Satisfying	3.58 (0.67)	3.3 (0.67)
Dull ----- Stimulating*	4.0 (0.85)	3.4 (0.84)
Difficult ----- Easy	3.5 (0.8)	3.4 (0.85)
Rigid ----- Flexible	3.83 (0.83)	3.4 (0.84)
Boring ----- Fun*	4.25 (0.45)	3.8 (0.92)

\* Difference approaches significance,  $p < .10$

A quick review of the means show that the users overall rated the new environment to be more promising than the older version. The mean differences were tested using a ANOVA; the test revealed that none of the difference were significant although the difference for ratings on Terrible-Wonderful, Dull-Stimulating, Boring-Fun approached significance ( $p < .10$ ).

The second set of questions used the Likert-scale to obtain user's reactions specific to the learning activities. Participants responded using a 5-point scale from 1=Strongly Disagree to 5=Strongly Agree. Most of the items were written with a positive

context such that a “5” would be a positive reaction; some items written with a negative context are noted using italicized text, and the ratings for these items have been recorded to be consistent with the others. Table 12 assess the general ease of use

Table 12. Likert-Scale Rating: General Ease of Use

Likert-Scale Rating	New Version Mean (SD) N=12	Old Version Mean (SD) N=10
Easy to learn and use	3.75 (0.87)	3.3 (0.67)
Easy to get started *	3.91 (0.67)	3.2 (0.92)
<i>Hard to remember tool locations</i> (NOT)	2.9 (1.24)	3.0 (0.94)
Easy for novices	3.58 (1.16)	3.3 (0.95)
I understand how to use **	4.16 (0.72)	3.7 (0.48)
<i>It was hard to recover from errors</i> (NOT)	3.66 (1.3)	3.1 (1.29)

\* indicates significance  $p < 0.05$ , \*\* indicates significance  $p < 0.1$

Examination of the above items pertaining to the general ease of use we see that in all aspects the mean values for the new version is higher than the older version. To assess the reliability of these raw differences, a simple ANOVA was conducted. These revealed that two of these mean differences were statically reliable. Easy to get started ( $p < .05$ ) and good understanding ( $p < .10$ ). This shows that the users found the new environment provided good understanding of the process and made it easy to get started with creating simulations.

Table 13. Likert-Scale Rating: Assessing Motivation

Likert-Scale Rating	New Version Mean (SD) N=12	Old Version Mean (SD) N=10
Fun for building simulations **	4.3 (0.49)	3.4 (1.07)
Creation of working simulation	4.25 (0.45)	3.7 (0.95)
I can have objects any size I want	4.08 (0.79)	3.7 (0.82)
I am enthusiastic about creating simulation	4.16 (0.83)	3.7 (0.82)

\*\* indicates borderline significance  $p < 0.01$

In all above cases under motivation as shown in Table 13, we note that the mean values of new version are higher than the previous version. To assess the reliability, simple ANOVA was conducted and found that the mean difference for the fun factor achieved a borderline significance of 0.01. So the new version has increased the fun during the exploration of the new environment increasing the motivation of the users.

Table 14. Likert-Scale Rating: Assessing Programming style Reactions

Likert-Scale Rating	New Version Mean (SD) N=12	Old Version Mean (SD) N=10
<i>Drag and drop rules were complicated</i> (NOT)	3.0 (1.28)	2.9 (0.99)
Simulation works logically but tools made it harder	2.83 (1.19)	2.7 (0.82)
Rule creation was simple and natural	3.83 (0.94)	3.7 (0.67)

An examination of the items designed to assess the programming style shown in (Table 14) that the mean difference is lower and ANOVA results revealed none of them were statistically significant. Thus we cannot conclude anything from the ratings.



To obtain the qualitative data, open-ended questions were provided to know the user's reactions. The following paragraphs summarize their responses.

*1. What was most interesting or fun?*

Users commented that it was interesting to create, erase and to redesign the objects. They had fun with adding scripts to the objects. Set of users were impressed the way the simulation worked as they intended. Many of them considered drawing the volcano simulation using the paint brush was more fun.

*2. What was least interesting or fun?*

Users commented some of the least interesting and fun things for them to do dealt with adding actions/behaviors to the objects, bringing the halo for each object by clicking the mouse and one among the 12 participant responded that drawing was least interesting as he/s she was not an artist.

*3. Did you find the example simulations in the tutorial effective? Why or why not?*

Everyone found that the example simulations used in the tutorial were effective for the following reasons: It provided good understanding of the process involved before starting the simulation; it helped me to get started easily. One participant suggested having the tutorial part on the computer rather than the hard copy.

*4. Did you find the instructions in the tutorial helpful? Why or why not?*

Everyone found that the instructions in the tutorial were helpful. Some of the comments were: Yes, it went step by step along with me with the diagrams explaining all the steps; yes, it was straight-forward; yes, they were clear and precise. One participant felt that not all the necessary steps for final simulation were detailed.

5. *What 1-2 things would you change if you were asked to revise the tutorial?*

Some of the remarks were as follows: Some of the instructions could be clear; More specific instructions; Provide detail information about certain menu items; Step by step tutorial within the program itself instead of power point slides.

6. *Suppose you were going to build a computer simulation of a volcano exploding for earth science. What sort of things do you think would be involved?*

Everyone mentioned the basic objects like lava, mountain, smoke, grass, clouds, rocks falling, and vibrations.

7. *As well as you can, please describe what you think is the best way to come up with projects?*

After being exposed to the environment, users commented the following: If its for science class , then any subject that cannot be re-created ordinarily in classroom can be made as simulation; Inculcate creativity; Understand the students weak subjects and do simulations in that for better understanding; Projects that cannot be viewed in the classroom ; simulation showing how caterpillars become butterflies.

8. *Can you think of any changes or enhancements to this system, especially ones that would make it more useful in creating simulation for novices?*

- More shapes for drawing.
- Include help session.
- Scripts should be more kid-friendly.
- Baseline objects necessary.
- Selecting halo can be made easier.

9. *Any final comments about your experiment or the software.*

- It was fun using the system.
- Young science students will find it more helpful.
- Extensive training is necessary.
- What I expected to happen I got it easily done with the system.
- It takes long time to get comfortable.
- Better than the previous Squeak version.

A retrospective interview was also conducted at last to give a chance for the participants to tell out what they felt about the environment. Most of the questions were similar to those listed above but in addition asked about the stylus to the participants who used it.

10. *Do you think a stylus was helpful for drawing application? How do you support and for what action it was helpful?*

Among the six participants, 5 of them responded that using stylus was helpful compared to the mouse. One participant told that after getting used to stylus, he/she found it more natural to draw like using a pencil. Few participants mentioned that stylus was too sensitive and so took time to get used to it. Only one participant felt that mouse was less complex than the stylus for drawing. Stylus was more preferred for drawing.

## CHAPTER 7 .DISCUSSIONS AND CONCLUSIONS

The general aim of this research was to increase the user satisfaction of the Squeak SimBuilder environment. It was initiated by conducting a usability study for identifying the difficulties the novices faced using the Squeak SimBuilder 3.7 version for creation of science simulations. Utilizing the Natural programming design process, those identified factors were used to motivate and guide the redesign of the environment to achieve effectiveness, efficiency and satisfaction of users. The usefulness and usability of the redesigned version was contrasted with the older version using empirical evaluations. The quantitative, and qualitative data collected during the study summarizes that in overall the redesigned version has number of advantages compared to previous version. This research also studied whether the use of stylus for interaction would increase the performance of users, making it more fun to learn and use.

### 7.1 Results Summary

The research addressed questions related to whether the newly redesigned visual environment has potential advantages over the existing environment. A set of four hypotheses were framed to support these questions. First we wanted to explore how easy the system to understand is. The novices when they investigate a new environment will try to explore only the high level features of it before creating working simulations. The time they require to understand the system and the important aspects they understand are

very crucial. We captured the time while the users practice the learning session, and compared it with the time obtained in the study with the earlier version. Though the time required with our interface did not show a significant difference, the Likert-Scale rating assessing how well the system provides good understanding was statistically significantly with  $p < 0.1$  for this hypothesis. Therefore, the hypothesis was accepted and we conclude that our redesigned system provides better understanding of the system helping novices to easily get started with the creation of working simulations.

Second, we noticed that excessive flaps and menus in the environment make it hard for the novices, to locate tools required for basic simulations. Users should not be overloaded with the information when they explore the new environment. The new environment was redesigned with fewer and necessary flaps after proper consideration. The hypothesis for this was that the new environment will make it easier to locate the tools thus reducing the confusions. A significant level of 0.05 was not obtained for this hypothesis. Therefore, the hypothesis was not supported.

Third, we were concerned about the interaction style, whether stylus would improve the flexibility of environment. Qualitative data was collected to test this hypothesis. Stylus was preferred for creations of simulations since it produced a more natural way of drawing and painting than the mouse. But we also found that highly sensitive stylus pad frustrated the users. We expect that stylus with less sensitive pad would enhance the user's interaction with the environment.

Finally, we wanted to analyze whether the users had fun working with the environment. Draper (2002) stated that fun is a candidate software requirement in design of any software where learning is the main function. Inculcating fun in software design

would help to achieve the learn ability easily. The hypothesis tested for this was that the new redesigned environment would be more fun to explore and build simulations. A significant level of 0.01 was obtained for this hypothesis. Therefore, the hypothesis was strongly supported.

## 7.2 Future Work

- In future studies we need a more detailed analysis of areas of reusable objects. Few categories of generic objects should be identified from the existing simulations, which can be used for the new simulations without any change in the scripts.
- The existing simulations are based on the earth and physical science models, which can be extended to the biological sciences.
- We can port Squeak SimBuilder 3.7 to PDA so that it will be handy to use and can be shared easily among students in the class thus helping them in handling their projects easily.
- It could prove more beneficial to re-conduct the study with a population of school students to increase the reliability of the word “novice programmers”.
- The tutorial, which is now in power point slides, can be provided in the Squeak environment itself.

## 7.3 Conclusion

The contribution of this research is simple. This research led to analyze the opportunities and limitations of the existing visual programming environments for educational simulations. After understanding the difficulties of novices in the

programming field, we aimed at redesigning an environment to make it more usable, easier-to-use, fun, and stimulating environment. We expect that this environment will increase the accessibility of programming systems and help novices to understand the programming concept easier. Our belief is that using visual tools as method of instruction for the students would be revolutionize the education system.

## REFERENCES

- Alan Dix, Janet Finlay, and Gregory Abowd.(1997). Human-computer Interaction. Prentice Hall.
- Alex Repenning. (1991). Creating User Interfaces with AgentSheets. Proceedings of the 1991 IEEE Symposium on Applied Computing, April 3-5,1991, Kansas City, Missouri, 191-196.
- Alexander Repenning and J. Ambach. (1996). Tactile Programming: A Unified Manipulation Paradigm Supporting Program Comprehension, Composition and Sharing. In 12th IEEE Symposium on Visual Languages, 102-109.
- Alexander Repenning and Andri Loannidou.(2004).Agent-Based End User Development. Communications of the ACM September 2004.
- Amy Bruckman and Alisa Bandlow.(2002).HCI for Kids.Published in The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging applications.
- Andrew Jensen Ko and Bob Utzl.(2003).Individual Differences in Program Comprehension Strategies in Unfamiliar Programming Systems. Proceedings of the 11th IEEE International Workshop on Program Comprehension, Portland, Oregon, May 05 – 10.
- Andrew J. Ko, Brad A. Myers, and Htet Aung (2004).Six Learning Barriers in End-User Programming Systems.IEEE Symposium on Visual Languages - Human Centric Computing, Rome, Italy. September 26 – 29.
- Brad A. Myers. (1986).Visual Programming, Programming by Example, and Program Visualization Taxonomy. ACM. Proceedings of the 1986 ACM SIGCHI Conference on Human Factors in Computing Systems.
- Brad A. Myers. (1996).A Brief History of Human Computer Interaction Technology. CMU-CS-96-163, CMU-HCII-96-103.
- Brad A. Myers (1998).Natural Programming: Project Overview and Proposal. CMU-CS-98-101, CMU-HCII-98-100.



Cheryl Seals, Mary Beth Rosson, John M. Carroll, Tracy Lewis, and Lenese Colson (2002). Fun Learning Stagecast Creator: An Exercise in Minimalism and Collaboration, IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC'02) 09 03 - 09, Arlington, Virginia, USA

Christopher Jones. (1998).The Mouse that Squeaked.  
<http://www.wired.com/news/technology/0,1282,16833,00.html>

Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay (1997).Back to the Future: The Story of Squeak, A Practical Smalltalk written in itself. In Proc. of the ACM SIGPLAN conference on Object-oriented programming systems, languages and applications, Atlanta, GA USA, October 1997.  
<ftp://st.cs.uiuc.edu/Smalltalk/squeak/docs/OOPSLA.Squeak.html>

Daniel Fallman. (2003). Design-oriented Human-Computer Interaction. Proceedings of the SIGCHI conference on Human factors in computing systems, Ft. Lauderdale, Florida, USA. April 5-10.

Dan Shafer.(1996).The Future of Squeak. The We Talk Network, Inc.

Draper, S.W. (2002, May 14). *Web page title* [WWW document]. Retrieved 2003 April 1. <http://www.psy.gla.ac.uk/~steve/fun.html>

Gould, J. D., & Lewis, C.(1985).Designing for usability: Key principles and what designers think.Communications of the ACM, 28, 3, 300-311

Guzdial, Mark and Rose, Kim (2002) Squeak: Open Personal Computing and Multimedia. Prentice Hall.

Hal Eden, Mike Einsenberg, Gerhard Fischer, and Alexander Repenning (1996).Making learning a Part of Life. Communications of the ACM April 1996/vol 39.No.4

Jacob Nielsen and R.L Mack. (1994). HeuristicEvaluation.Usability Inspection Methods. New York, John Wiley & Sons: 25-62.

James A. Landay and Brad A. Myers (2001).Sketching Interfaces: Towards More Human Interface Design. Proceedings of IEEE.

James C. Spohrer and Elliot Soloway (1986).Novice Mistakes: Are the folk wisdoms correct?. Communications of the ACM, July 1986, Volume 29, No 7.

Jan Erik Moström. (2002).Using Concurrent constructs in an Authoring Environment.

Jim Gindling, Andri Ioannidou, Jennifer Loh, Olav Lokkebo, Alexander Repenning. (1995). LEGOsheets Rule-Based Programming, Simulation and Manipulation Environment for the LEGO Programmable Brick. Proceedings of the 11th International IEEE Symposium on Visual Languages.

John F.Pane, Chotirat “Ann” Ratanamahatana and Brad A. Myers. (2001).Studying the language and structure in non-programmers’ solutions to programming problem. Natural Programming project, CMU.

Joseph S. Dumas and Janice C. Reddish. (1993).A Practical Guide to Usability Testing. Alex Publishing Corporation, Norwood, New Jersey.

Jurgen Herzeg, Hubertus Hohl, and Matthias Ressel.(1993).A New Approach to Visual Programming in User Interface Design. Proceedings of HCI 1993, Orlando, FL, USA.

J.W. van Aalst, C.A.P.G. van der Mast, and T.T. Carey. (1995). A Multimedia tutorial for User Interface Design. Report 95-35.

K.A. Whitley and Alan L. Blackwell. (1997).Visual Programming: The Outlook from Academia and Industry. Paper presented at the seventh workshop on Empirical studies of programmers.

Kori Inkpen. (1997).Three Important Research Agendas for Educational Multimedia: Learning, Children, and Gender. Proceedings of Educational Multimedia '97, Calgary, AB, June 1997, pp.521-526.

Margaret M. Burnett, Marla J. Baker, Carisa Bohus, Paul Carlson, Sherry Yang, Pieter van Zee (1995).Scaling Up Visual Programming Languages .Proceedings of IEEE, Volume 28 , No.3.

Martijn van welie, Gerrit C.van der Veer, Anton Eliens. (2000). Patterns as Tools for User Interface Design.Virje University.

Mary Beth Rosson, John M.Carroll, and Rachel K.E. Bellamy. (1990).Smalltalk scaffolding: A Case study of Minimalist Instruction.Proceedings of CHI 1990.

Mathew Conway, Steve Audia, Tommy Burnette, Dennis Cosgrove, Kevin Christiansen, Rob Deline, Jim Durbin,Rich Gossweiler, Shuchi Koga, Chris Long, Beth Mallory, Steve Miale, Kristen Monkaitis, James Patten, Jeff Pierce, Joe Shochet, Brian Stearns, Richard Stoakley, Chris Sturgill, John Viega, Jeff White, George Williams, Randy Pausch. (2000). Alice: Lessons Learned from Building a 3D System for Novices. Proceedings of CHI 2000, 486-493.

Ned Konz. (2004). A More Inclusive Community-Based Model for Squeak Development. <http://bike-nomad.com/squeak/community2.html>

N.Hari Narayanana and Roland Hubscher. (1997). Visual Language Theory: Towards a Human-Computer Interaction Perspective in *Visual Language Theory*, pages 85-127, Springer Verlag, New York, NY, 1998.

Olivier Esteban, Stephane Chatty, Philippe Palanque. (1995). Whizz'Ed: A Visual Environment for Building Highly Interactive software. In K. Nordby, P. Helmersen, D.J. Gilmore & S.A. Arnesen (Eds.), *Human Computer Interaction: Interact '95*. London: Chapman & Hall, pp. 121-126.

Randall Caton. (2004). Squeak Interactive Web Activities Developed for the NASA Center for Distance Learning. NASA Langley Research center.

Richard E. Mayer. (1981). The Psychology of How Novices Learn Computer Programming. *Computing Surveys*, Vol.13, No.1

Silvia Berti, Fabio Paterno, and Carmen Santoro. (2004). Natural Development of Ubiquitous Interfaces. *Communications of the ACM* September 2004/vol 47.No.9

Travers, M. (1994). Recursive Interface for Reactive Objects. *Proceedings of CHI'94*, in Boston, Massachusetts.

T.R.G. Green and M. Petre. (1992). When Visual Programs are harder to Read than Textual Programs. *Human-Computer Interaction: Tasks and Organization*, *Proceedings of ECCE-6 (6th European Conference on Cognitive Ergonomics)*. G. C. van der Veer, M. J. Tauber, S. Bagnarola and M. Antavolits. Rome, CUD.

Vikki Fix, Susan Wiedenbeck, Jean Scholtz. (1993). Mental Representations of Programs by Novices and Experts. *Proceedings of the SIGCHI conference on Human factors in computing systems, INTERCHI '93*, Amsterdam, The Netherlands, pp.74 – 79.

## APPENDICES

APPENDIX A: INSTITUTIONAL REVIEW BOARD FORMS: INFORMED CONSENT .....	90
APPENDIX B: EVALUATOR MONOLOGUE.....	92
APPENDIX C: PRE-QUESTIONNAIRE .....	93
APPENDIX D: SIMBUILDER TUTORIAL .....	98
APPENDIX E: POST-QUESTIONNAIRE.....	107
APPENDIX F: STATISTICAL ANALYSIS .....	110
APPENDIX G: RETROSPECTIVE INTERVIEW QUESTIONS.....	114
APPENDIX H: CREATION AND REUSE TIMES .....	116



Visual Programming 2005

INFORMATION CONSENT SHEET

for Research Study Entitled

SimBuilder: An Investigation and Usability study of Novice Programming Techniques

You are invited to participate in a research study which aims in studying the problems faced by the novice programmers when using a visual programming tool called Squeak. The usability of our new design will be evaluated against the latest versions of the software available in the market. This study is being conducted by Dr. Cheryl D. Seals, Assistant Professor and Sumitha Kanakadoss, graduate student of Computer Science and Software engineering Department. We plan to analyze the difficulties of novices and make programming easier and fun with our new design thus decreasing the learning curve. You were selected as a participant because you are computer literate, enrolled in a computer science graduate or undergraduate course.

If you decide to participate, you should be able to spend 1 hour for this entire study. First you will take a pre survey which will provide us some background information about you. After that a learning session of 30 minutes will be provided to get familiarized with the environment. Once you are comfortable with the environment, the next stage is that you will be asked to create a simulation of your own. A post survey will be done at the end of this to understand the user's reaction towards this environment.

Any information obtained in connection with this study will remain anonymous. Information collected through your participation may be used to fulfill an educational requirement (Thesis), published in a professional journal, and/or presented at a professional meeting.

While there are no direct benefits to you from this research, you may find the research and interaction with the new educational tool interesting. Your participation should make it possible to better understand the opportunities provided by a visual programming tool like squeak.

Your decision whether or not to participate will not jeopardize your future relations with Auburn University or Computer Science and Software engineering Department. You are free to withdraw from this study at any time without any question.

If you have any questions we invite you to ask them now. If you have questions later, you can contact either Dr. Cheryl D. Seals ([sealscd@eng.auburn.edu](mailto:sealscd@eng.auburn.edu)) or Sumitha



## Appendix B: Evaluator Monologue

### EVALUATOR MONOLOGUE

1. Complete the pre-survey

2. Learning Session

- Follow the steps given in the tutorial properly.
- Explore the Water cycle simulation.
- Create few new rules to get familiarized with the Environment.
- ‘Think aloud’ while you are learning about the new environment.
- Make use of the interaction guidelines provided at the end of tutorial.

3. Creation

- A sample environment of volcano eruption is provided.
- Identify the various objects involved in it and their interaction.
- Draw those and gives the rules for their behavior.

4. Reuse

- Explore the Starter/Ozone world which is provided for you in the main page.
- Create the Ocean/Photosynthesis (evaluator will tell you which one to do) model reusing the objects.

5. Complete the post-survey

6. Retrospective Interview

Appendix C: Pre-Questionnaire

Visual Programming Summer 2005

Sim ID

Age

Gender

Major

Educational background: Please list any degrees or courses taken in the following areas.

Software Design

Computer Programming

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 or pocket pc?

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..

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



# Exploring Visual Programming

Squeak SimBuilder Tutorial  
Implementing a Model of the Water Cycle

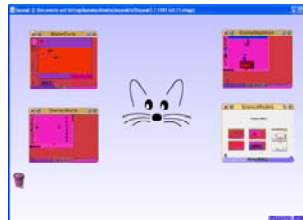
Human Computer Interaction  
@  
Auburn University

This tutorial is a draft of materials being developed as part of behavioral research underway in the CSSE at Auburn. It is provided on an "as-is" basis; however, we welcome comments and suggestions. Please direct any feedback to [sealscd@auburn.edu](mailto:sealscd@auburn.edu).

*SimBuilder Tutorial © AU Computer Human Interaction Laboratory*


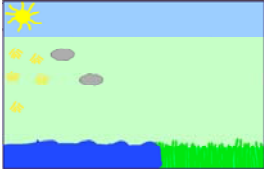


## Preliminaries

- The purpose of this document is to introduce you to SimBuilder by helping you create a small project.
- SimBuilder is designed for building simulations, such as a model of the water cycle.



- No programming experience is required.
- In this experiment we want to evaluate the usefulness of this tool to a science teacher to simulate environments or lab experiments as curricula aids in his/her classroom with SimBuilder.
- The last page of this document contains an interaction guide.

## Exploring the Water Cycle Model


- **Double click the**  **icon on the desktop to open the Squeak environment.**  
Squeak
- **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.


## Exploring The Sun

### Let's investigate the players


To investigate the **Sun**

- **<Alt> <Left button>** or **<Middle button>** Click the **Sun** and it's **Halo** will appear.

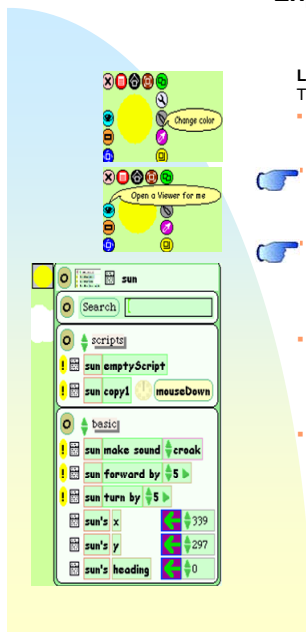
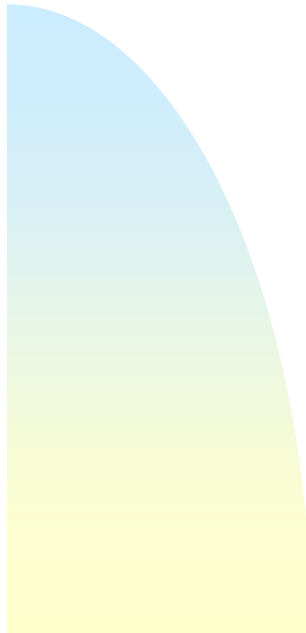
 Select the grey **Repaint handle** tool to change the **Sun** from yellow to bright orange.

 To investigate the rules or **scripts** that govern the sun's behavior, Select the blue eyeball to **Open a Viewer of me**. This will show you the set of graphical rules for this player.

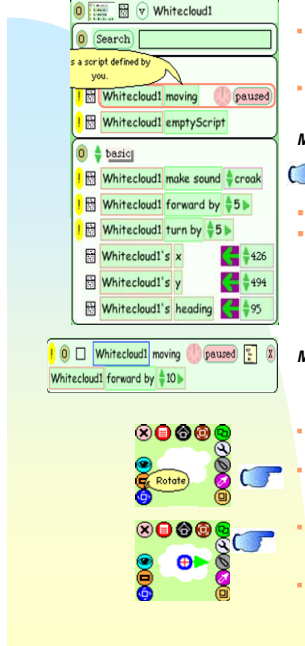
- What does the **Sun** do?

- You program by simply selecting a rule, then dragging and dropping the  **scripts** out of the scripting window and placing it somewhere in the world other than the green playground.

*(Take a few minutes to explore scripts in the viewer...)*



## Changing The Behavior of White Cloud



- Examine some of the more complicated behaviors by selecting the White **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 White Cloud more active.

Currently the behavior of the white 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 "moving" from the **scripts** and move it outside the playfield
- Increase the value to 10 for moving forward.

- 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> <Left button>** or **<Middle button>** Click your cloud and it's **Halo** will appear.
- Select **Rotate** and move your cloud just a tiny bit for it's direction arrow to appear.



In order to change the direction that the **player** moves Click on the **green arrow** and drag it until it points up.

- Press **go** to see how the simulation has changed.

## Creating a Bird

A key aspect of creating new simulations is to build new agents. We will begin with the small task of adding a new agent to the Water Cycle project.

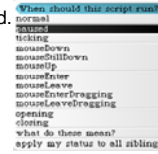
### Drawing a bird

- Select **draw** from File menu 
- Use a brush and the color palette to draw something that looks like a bird.
  - Refer to the interaction guide for **Gallery Tools** if you need to refresh your memory. *This is just for fun. Do not worry if you are not a good artist.*
- Once finished Press **Keep** 
- Your **bird** has been added to the playground. It's just as easy as that !!!

## Giving the Bird Behaviors

In this playground, we want the bird to be able to fly through the sky.

- **<Alt> <Left button> or <Middle button>** Click your bird and its **Halo** will appear and open its viewer.
  - ◆ The **viewer** is a window to select scripts for your object.
  - ◆ Let's add the behavior that will make your agent move in one direction.
  - ◆ Select click on normal to change it to paused.



and drag it out of the **viewer** and place it outside the playground and change **normal** to **paused**.

- **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> <Left button> or <Middle button>** Click your bird and its **Halo** will appear.
- **Select Rotate and move your bird just a tiny bit for it's 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!!**

## 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/players for your volcano simulation.

*(Take 2-5 minutes brainstorming and drawing.)*





## Creating a Volcano Simulation

- To Leave WaterCycle  
Select File and Press <PREV.



- Now that you are back in the Welcome page.  
Select File and Press NEW project
- 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 your  in the **File** tab to paint whatever you like.


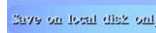
## Adding Behaviors to Volcano Simulation


- 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.
- To make your volcano erupt, the **players** need to interact with each other. To 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 agents to collaborate in interesting ways.

**When finished Save your project.**

Press  in the File and then 

Press  project and you should be back at the Welcome page.

# Exploring Visual Programming

---

## Section II

### Reusing Objects 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)*

## Reusing the Objects



Starter World

From the main page click the **Starter World** and hit **go** to see what happens.

There are a set of objects provided in the **Objects** Flap which can be reused for other simulations.

- Investigate each player to discover its' behavior.



The Mover just moves in one direction.

The Emitter produces another agent.

The Eraser erases other agents that it contacts.

The Replacer replaces the Mover with another agent.

The Changer will change another player into a new player when it comes in contact with it.

- Refer to interactions guide for **Help**.

## Creating new agents 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 reusing existing players and adding new agents. We will begin with the small task of reusing a new **player** from the Objects flap.

### Task 1. Create a new project

- In the File tab 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. **new players**

Refer to interactions guide for **Help**.

### Task 3. Reusing and creating new behaviors for your new player

- Some player already have behavior scripts. You may need to look at their behaviors to get started.
- To 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.

## Reusing Ozone World



Ozone Depletion

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 a more active form. No ozone loss occurs until sunlight initiates the catalytic ozone destruction.

Open the Ozone Depletion Simulation in the main page.

- Now, **Press**  **to start** 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**.



## Interaction Guide (Object Halo & Handles & Paint Tools)



### Halo Tools



To manipulate objects in Squeak *SimBuilder* select the object and **<Alt>< Left button>Click** or **<Middle button> Click** your object and it's **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.



## Interaction Guide (Object Behavior & Scripting Tools)



### Scripting Tools

To open more tools click

### Saving your Work

To Save your projects.

- Press **PUBLISH IT!** and
- then **Save on local disk only**

### Scripts define behaviors



A user defines the behavior of objects by creating a script for that object.

To use one of the predefined scripts. **Select a script** and **drag** it from the script window into the world any place other than the playground.

Press **!** to **Test** an individual script.

### Scripting Categories

The **Search** area allows you to quickly locate a script with the search.

The **scripts** category is where user created scripts are located.

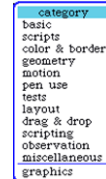
The **basic** category is to make sounds and move your object.

The **color & border** category is to make sounds and move your object.

The **tests** category contains scripts that help you to test conditions.

The **graphics** category is to make sounds and move your object.

The **miscellaneous** category contains many scripts (copy, show, hide, delete, etc. )



Appendix E: Post-Questionnaire

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:

This system was easy for me to learn and use.

Strongly Agree  Agree  Neutral  Disagree  Strongly Disagree

It was easy to get started.

Strongly Agree  Agree  Neutral  Disagree  Strongly Disagree

It was difficult to remember where some of the tools and commands were located.

Strongly Agree  Agree  Neutral  Disagree  Strongly Disagree

This system would be easy to use by folks who don't know much about computers.

Strongly Agree  Agree  Neutral  Disagree  Strongly Disagree

This system would be fun for building simulations.

Strongly Agree  Agree  Neutral  Disagree  Strongly Disagree

I have a good understanding of how to use this system to build simulations.

Strongly Agree  Agree  Neutral  Disagree  Strongly Disagree

I was able to use this system to turn my ideas into working simulations.

Strongly Agree  Agree  Neutral  Disagree  Strongly Disagree

Creating visual rules by dragging and dropping the desired parts to create behavior was complicated.

Strongly Agree  Agree  Neutral  Disagree  Strongly Disagree

My simulation works logically, but the tools made it hard to create the desired behavior

Strongly Agree  Agree  Neutral  Disagree  Strongly Disagree

It was hard to recover from errors.

Strongly Agree  Agree  Neutral  Disagree  Strongly Disagree

The rules I created for objects' behaviors were simple and natural.

Strongly Agree  Agree  Neutral  Disagree  Strongly Disagree

I was able to have agents any size I wanted.

Strongly Agree  Agree  Neutral  Disagree  Strongly Disagree

At this point, I am enthusiastic about creating new simulations.

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?)

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 F: Statistical Analysis

Table 15 . Anova 0.1

Anova: Single

Factor

Terrible -Wonderful

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
Summer05	12	48	4	0.545455
Spring 05	10	35	3.5	0.5

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	1.363636	1	1.363636	2.597403	0.122709	4.351243
Within Groups	10.5	20	0.525			
Total	11.86364	21				

Table 16 . Anova 0.2

Anova: Single

Factor

Dull -Stimulating

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
Summer05	12	48	4	0.727273
Spring 05	10	34	3.4	0.711111

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	1.963636	1	1.963636	2.727273	0.114263	4.351243
Within Groups	14.4	20	0.72			
Total	16.36364	21				

Table 17 . Anova 0.3

Anova: Single  
Factor Boring -fun

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
Summer05	12	51	4.25	0.204545
Spring 05	10	38	3.8	0.844444

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	1.104545	1	1.104545	2.242732	0.149861	4.351243
Within Groups	9.85	20	0.4925			
Total	10.95455	21				

Table 18 . Anova 0.4

Anova: Single  
Factor Fun for building simulations

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
Column 1	12	52	4.333333	0.242424
Column 2	10	34	3.4	1.155556

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	4.751515	1	4.751515	7.272727	0.013875	4.351243
Within Groups	13.06667	20	0.653333			
Total	17.81818	21				

Table 19 . Anova 0.5

Anova: Single

Factor

Good understanding

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
Column 1	12	50	4.166667	0.515152
Column 2	10	37	3.7	0.233333

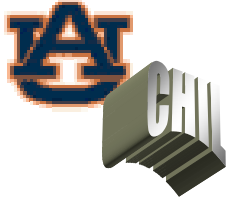
ANOVA

<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	1.187879	1	1.187879	3.058915	0.095627	4.351243
Within Groups	7.766667	20	0.388333			
Total	8.954545	21				

Table 20 . ANOVA results

Usability aspects	Mean Value (SD)		ANOVA	
	Summer 05	Spring 05	F(df)	p
System is wonderful	4(.74)	3.5(.71)	2.59(1)	.12
Satisfaction	3.58(.67)	3.3(.67)	.97(1)	.33
System is stimulating	4(.85)	3.4(.84)	2.72(1)	.11
Easy	3.5(.8)	3.5(.85)	0(1)	1
Flexible	3.83(.83)	3.4(.84)	1.45(1)	.24
Fun	4.25(.45)	3.8(.92)	2.24(1)	.14
Easy to learn and use	3.75(.87)	3.3(.67)	1.78(1)	.19
Easy to get started	3.91(.67)	3.2(.92)	4.47(1)	.04
Difficult to locate tools	2.9(1.24)	3(.94)	.03(1)	.86
System would be easy for students who know about computers	3.58(1.16)	3.3(.95)	.38(1)	.54
Fun for building simulations	4.3(.49)	3.4(1.07)	7.27(1)	0.01
Good Understanding	4.16(.72)	3.7(.48)	3.05(1)	.09
Turning ideas to working simulations	4.25(.45)	3.7(.95)	3.18(1)	.89
Difficult to create rules by drag and drop	3(1.28)	2.9(.99)	0.04(1)	.84
Tools make it hard to create behavior	2.83(1.19)	2.7(.82)	0.89(1)	.76
Hard to recover from errors	3.66(1.3)	3.1(1.29)	1.04(1)	.31
Rules are simple and natural	3.83(.94)	3.7(.67)	.14(1)	.71
Easy to create agents of any size	4.08(.79)	3.7(.82)	1.23(1)	.28
Enthusiastic about creating new simulations	4.16(.83)	3.7(.82)	1.72(1)	.20

## Appendix G: Retrospective Interview Questions



Visual Programming 2005

Interview Questions

ID \_\_\_\_\_

1. What other ideas do you have for simulation ideas?
2. Which was more enjoyable drawing or making the simulation work?
3. What things were the hardest for you to accomplish? What were the easiest?
4. Would you consider using this environment to train students to program or in your classroom if you taught introductory visual programming?
5. Do you feel that young students would be motivated to use this environment?
6. What support would you need to utilize this software in an introductory class?

7. Do you think a stylus was helpful for drawing application? Yes/No
  
8. How do you support your choice for or against using stylus?
  
  
9. Was the stylus useful for the following?
  - A. General Interaction
  
  - B. Creation of Rules
  
  - C. Selection of Objects.

Appendix H: Creation and Reuse Times

PID	Learning Time(min)	Creation Time(min)	Reuse Time(min)	Reuse Method	Interaction Style
1	10	12	10	OZ->Ph	Mouse
2	17	20	18	St->Oc	Mouse
3	12	18	16	OZ->Ph	Mouse
4	17	18	12	St->Oc	Stylus
5	10	18	12	St->Oc	Stylus
6	24	15	39	St->Oc	Stylus
7	13	15	8	OZ->Ph	Stylus
8	46	17	30	OZ->Ph	Stylus
9	8	5	16	OZ->Ph	Mouse
10	13	15	12	St->Oc	Mouse
11	15	11	18	OZ->Ph	Stylus
12	8	9	11	St->Oc	Mouse
Average(min)	16.08333	14.41667	16.83333		
Stdev(min)	10.44865	4.399552	9.033607		

