NETSCRATCH: A NETWORKED PROGRAMMING ENVIRONMENT FOR CHILDREN

by

Tamara I. Stern

Submitted to the Department of Electrical Engineering and Computer Science

in Partial fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May, 2007
[June 2007]

Author _____
Department of Electrical Engineering and Computer Science
May 25, 2007

Certified by _____
Mitchel Resnick
LEGO Papert Professor of Learning Research, MIT Media Lab
Thesis Supervisor

Accepted by _____
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee of Graduate Theses

NETSCRATCH: A NETWORKED PROGRAMMING ENVIRONMENT FOR CHILDREN

by

Tamara I. Stern

Submitted to the
Department of Electrical Engineering and Computer Science
May 25, 2007
In Partial fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

This thesis introduces NetScratch, a programming environment that enables children to make dynamic digital creations that interact across networks. The work is developed as an extension to Scratch, a multi-media programming environment designed for children to create interactive animations, art, games, and other dynamic creations. Particularly, NetScratch adds *websensors*, a tool to bring information from the web into a Scratch project, and *shariables*, a way to share information among projects. These features provide children with the tools to create their own web mashups, experiment with real-time changing data, and connect projects to each other. Using NetScratch, children can create personally meaningful networked projects, while learning important computational and design concepts. And, in designing these creations, children can think about how they want to interact across networks and how their creations affect their friends and communities.

Thesis Supervisor: Mitchel Resnick
Title: LEGO Papert Professor of Learning Research, MIT Media Lab

ACKNOWLEDGEMENTS

I would like to thank the people who made my time in the Lifelong Kindergarten group so enjoyable, meaningful, playful, and educational.

First, I would like to thank Mitchel Resnick, who has been my official and unofficial advisor for the past five years. It is not too far-fetched to say that he shaped my entire MIT experience, starting as my freshman advisor all the way through to my Master of Engineering thesis advisor. He has always been such a great source of inspiration, support, and care. What I have learned from him has had such a profound effect on me and I know it will continue to act as a guide for me in the future.

I would also like to thank my mentor, John Maloney, who is one of the most patient, kind, and supportive people I know. He spent so much time throughout my years at the Media Lab teaching me everything I know about Squeak, reassuring me he wouldn't leave the night before my user study until all network issues were debugged, reading over multiple drafts of this thesis, and so much more. He has been my mentor through this whole process and I am so grateful for him.

I want to thank the Lifelong Kindergarten group at the Media Lab for letting me be a part of such an inspiring community of wonderful people. I've learned so much from each of the members of the group and many of the ideas in this thesis were sparked by discussions with them. Specifically, I want to thank Natalie Rusk for always listening to me and supporting me, Jay Silver for the many discussions we had about this thesis, Andrés Monroy Hernandez for being the group's web 2.0 guru and inspiring the initial excitement about shared data and web mashups, Amon Millner for the constant encouragement and creative insight (including coming up with the term *shariables*), Evelyn Eastmond for leaving large footsteps for me to follow in as my LLK MEng predecessor, Chris Garrity for lending me her son and his friends for the NetScratch workshop, and Stephanie Gayle for always being there to help with everything. Also, thanks to Leo Burd, Robbin Chapman, Lis Sylvan, Oren Zuckerman, Rachel Garber, and Robbie Berg for being such great role models for me during my time in LLK.

I would like to thank the Electrical Engineering and Computer Science department, especially the course 6 academic administrator, Anne Hunter, and my academic advisor, Professor Rob Miller, for clarifying requirements, taking care of administrative details, and always reminding me that I'd survive this place.

Thanks to Rob Doherty from the MIT Writing Center who read through this thesis with me week after week and helped me make sure that I was effectively translating the thoughts in my head to words on this paper.

And lastly, I would like to thank my family and friends, without whom I would probably have gone crazy. They have been so supportive of me throughout my time in LLK, from printing out my Scratch projects, to reading blogs about what I've been up to in LLK, to trying to understand me as I spoke about my time here in what may as well have been another language (Scratch, Squeak, UROP, MEng, LLK, Cube, NetScratch, *websensors*, *shariables*, etc.), to helping me find apartments in New York so that I could focus on my work here in Boston, and everything in between. Thanks so much, especially mom, dad, Steven, and Harel, for everything.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 CHILDREN AND NETWORKS

Computer networks are used in a multitude of educational contexts. Some educators see networks as an avenue to deliver information to children, through online lectures, tutorials, and educational games, with children acting as passive consumers. Others see networks as a tool for children to actively seek out information that is meaningful to them, using search engines, blogs, and news reports. Yet another role networks play in children's lives is as a communication device, providing children with a way to connect and collaborate. Mitchel Resnick, LEGO Professor of Learning Research at the MIT Media Lab, views networks in a different way. He sees them "not as a channel for information distribution, but primarily as a new medium for construction, providing new ways for students to learn through construction activities by embedding the activities within a community (1)."

Like Resnick, I believe networks go beyond channeling information; they can actually serve as a space to nurture a child's creativity and learning process. With the right tools, children can move beyond the usual consuming, browsing, and interacting on networks, and move toward designing, inventing, and sharing on networks.

This thesis introduces NetScratch, which is designed to enable children to create interactive media in a networked context. NetScratch is an extension to Scratch (2), a graphical programming environment developed by the Lifelong Kindergarten group (3) at the MIT Media Lab (4). Using NetScratch, children can create personally meaningful networked projects, while learning important computational and design concepts. And, in designing these creations, children can think about how they want to interact across networks and how their creations affect their friends and their communities.

1.2 CHILDREN AND PROGRAMMING

Years of research have shown that children learn best when actively involved in designing and creating their own inventions. Based on these ideas, Seymour Papert (5) developed a theory called *Constructionism*, suggesting that learning is an active process in which learners construct mental models

and theories by actively designing and constructing things in the physical world. Papert likened these constructionist ideas to his experience with gears in his childhood; because he loved gears growing up, he was able to easily understand them and bring that understanding to other aspects of his learning. Papert believed that computers, like the gears, could provide children with the space to explore and think in new and personal ways and then bring those types of thinking to other aspects of their learning. In the 1960's, Papert and his research team developed the Logo programming language, with the hopes that children would learn by teaching, or programming, computers how to behave.

Following in the spirit of Papert and the Logo traditions, the Lifelong Kindergarten group in the MIT Media Lab, led by Mitchel Resnick, specializes in designing new technologies that expand the range of what people can design, create, and learn. Specifically, the Lifelong Kindergarten group aims to introduce new technologies and activities in classrooms, museums, and after-school centers, thereby nurturing and encouraging the study of communities composed of playfully-inventive learners.

One of the efforts of the Lifelong Kindergarten group is an international network of after-school centers called Computer Clubhouses. This effort, funded by Intel and run in conjunction with the Boston Museum of Science, is a worldwide endeavor to bring technology to underserved communities who would not otherwise have access to these technological tools and activities. The culture of the Computer Clubhouse is one of self-motivation, as children choose to work on projects in which they have a personal interest.

Many informal cultures have developed in Computer Clubhouses. For example, a visual-design culture has emerged where children creatively express themselves with professional graphic-design and image-processing software. Clubhouses have also developed a thriving music-production culture.

These cultures are very productive in the Clubhouse, but they do not engage children in programming as a form of self expression. In the Lifelong Kindergarten group, we believe that the ability to program provides children with expanded opportunities to express themselves, to create projects they care about, and to understand core computational ideas and concepts. In response to these beliefs, the group decided to build Scratch, with the goal of introducing a "programming culture" at Computer Clubhouses (6).

Scratch is a programming environment, built upon the Squeak programming language (7). Scratch builds on the tradition of other educational programming languages, such as Seymour Papert's (8) Logo and Alan Kay's (9) Etoys, but takes advantage of new computational ideas to make it easier to get started with programming and to extend the range of what children can create and learn.

Using Scratch, children can create their own interactive animations, videogames, newsletters, stories, music, simulations, and art. In designing and developing Scratch projects, children learn mathematical, computational, and design concepts in a meaningful and motivating context.

1.3 Children Programming in Networks

Although programming as a form of self expression has been successful in some contexts, it can often be an individual and isolating process. As a result, programming has not lived up to its potential as a means for creative self-expression among children. Networks offer a new pathway into programming, enabling children to construct programs collaboratively and to think about their programs in the context of a community. In particular, creating within a community setting offers an audience and inspiration.

Creating for an audience serves a few purposes. First, it provides motivation for the creator who knows his or her work will be available for others to see. In their analysis of programming environments for novice users, Kelleher and Pausch claim there are sociological barriers to programming, including the lack of a social context (10). By programming on a network, the inventor is now creating in a community, expressing him or herself in a public way that can affect people. Such a context enables children to think about their friends and their community and how they want to affect the people with whom they interact through networks.

An audience also provides a space for back-and-forth collaboration and feedback. When a child is creating with or for someone, there is a space for a variety of suggestions for how to go about solving a problem, how to improve a particular problem-solving approach, or how to think about the problem in a completely new way. Drawing on research in *distributed cognition* (11), it is believed that "cognition and intelligence are not properties of an individual person but rather arise from interactions of a person with the surrounding environment (including other people and artifacts) (1)." Thus, networks can

enhance learning by creating a fostering "surrounding environment," in which children work together to solve problems that they care about.

Networks also provide children with a space where they can be inspired. Online, children can see other people's final projects and works in progress, providing a glimpse into their thought processes and ideas. Networks are a space where children can be inspired to come up with their own ideas based on the creations of others.

1.4 NetScratch, A Networked Programming Environment

This thesis introduces NetScratch, an extension to the Scratch programming language. NetScratch combines the benefits outlined in the previous sections, enabling children to program personal and meaningful projects in a collaborative and networked context.

Specifically, NetScratch introduces *websensors*, a means to bring information from websites into a Scratch project, and *shariables*, a means to share information among projects. Using these features, NetScratch users can connect and collaborate by using networked information. For example, the NetScratch project presented in Figure 1 connects three NetScratch users across the country enabling them to dynamically update their feelings and chat with one another using *shariables*. Additionally, they are able to visualize each friend's location and associated weather conditions using *websensors*.

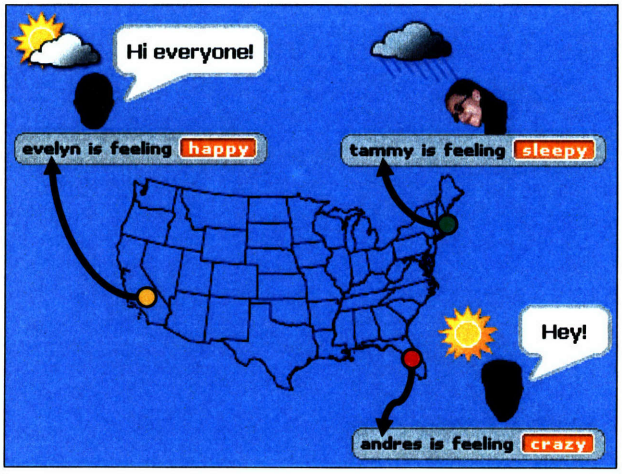


Figure 1: Networked project created using NetScratch

NetScratch not only enables children to design and create their own networked projects, but also exposes them to important ideas, such as the design process, distributed computing, and real-time data. Like Resnick and Silverman, inventors of countless creative toolkits, my goal is to "not only engage kids in constructing things, but also encourage (and support) them to explore the ideas underlying their constructions (12)." Using NetScratch, children are not only engaged in creating their own online games, chat systems, and other expressive creations, but are also encouraged to think about how these prevalent technologies are carefully designed and developed.

1.5 Thesis Overview

This thesis presents the motivation for building NetScratch, the NetScratch environment and capabilities, and an analysis of NetScratch based on user feedback. Particularly, the thesis is organized in the following way. Chapter 2 presents other programming environments that leverage networks. Chapter 3 discusses the current means for dynamic interaction and collaboration in Scratch, a programming environment for children. Chapters 4-6 present the design, usage scenarios, and implementation of NetScratch. Chapter 7 presents user study results. Finally, Chapter 8 concludes and offers recommendations for future work.

2 Related Work

Currently several other programming languages aimed at novices have made attempts to leverage networks in their environments. Although these other programs might have differing goals from NetScratch, and their models might not be fully applicable, there are lessons to be learned by looking at them. These applications served as informative examples to which I referred when integrating networked components into Scratch.

2.1 Etoys NetMorphs

Etoys is a scripting environment developed by Alan Kay's (9) research team. Like Scratch, Etoys enables users to manipulate objects interactively and intuitively to create meaningful projects. NetMorphs extends Etoys by enabling easy transfer of these scriptable objects across networks.

The goal of NetMorphs is similar to that of NetScratch; using the system "even beginners can develop distributed applications in a highly intuitive way (13)." NetMorphs carry out this goal by enabling objects in Etoys to migrate from one desktop to another, as shown in Figure 2.

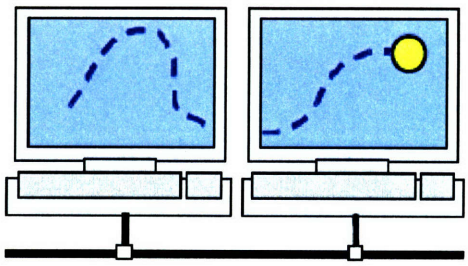


Figure 2: NetMorphs enable objects to seamlessly migrate from one machine to another

Supporting networked projects in this way enables children to seamlessly send active objects from one computer to another, much like mail or chat systems. Additionally, the original creator of an object can continue to control it as it moves to other computers. For example, one user might create a car with a steering wheel and drive it around on another user's screen.

NetScratch takes a slightly different approach. Instead of sending objects between machines, control over objects is shared via shared variables. This approach allows components of projects to exist in multiple projects at the same time, thus enabling multi-user real-time interactions.

2.2 AGENTALK

AgenTalk (14), the end-user derivative of Agentsheets, is a graphical programming environment in which conditions, actions, rules, and triggers are dragged and dropped to create programs. These programs control agents, which can move, change their appearance, open URLs, play sounds, read web pages, speak, receive mouse events, and poll the keyboard. A sample program is shown in Figure 3.

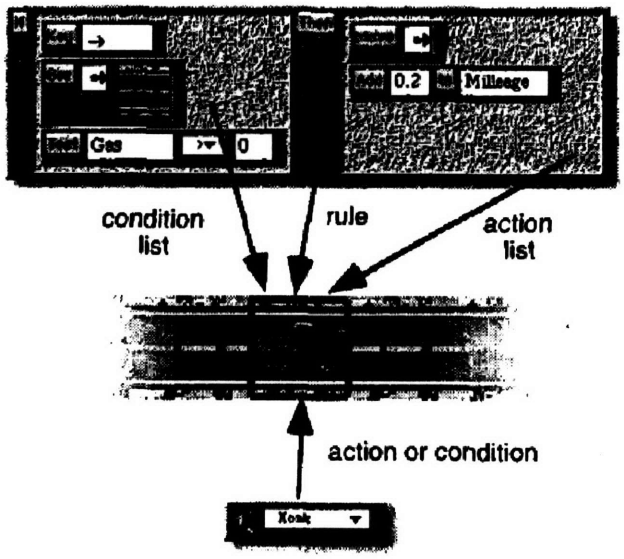


Figure 3: AgenTalk programs are created by dragging and dropping language components

Like NetScratch, AgenTalk was developed with the belief that programming in a social context enables novices to get started with self-expression through programming: "Programming approaches that are usable and expressive at the same time require reconceptualizing the programming process from a solitary solitaire activity to a social process that is supported with new kinds of tools (14)."

To support this idea, AgenTalk enables connecting to the web to retrieve real-time data. Additionally, AgenTalk enables the networked transfer of entire programs or components of programs. However, like

Etoys NetMorphs discussed in the previous section, once a program component is sent, various users cannot dynamically interact with it; it only lives on the machine to which it was sent. This transfer of program components, as opposed to real-time sharing, does not enable collaborative multi-user interactions.

2.3 ISLAY

Islay (15) is a tool for creating network-based interactive animations, particularly for presenting information on the web with which users can interact through mouse and keyboard inputs. These interactive animation programs consist of a collection of characters whose behaviors are specified by state-transition diagrams, as shown in Figure 4.

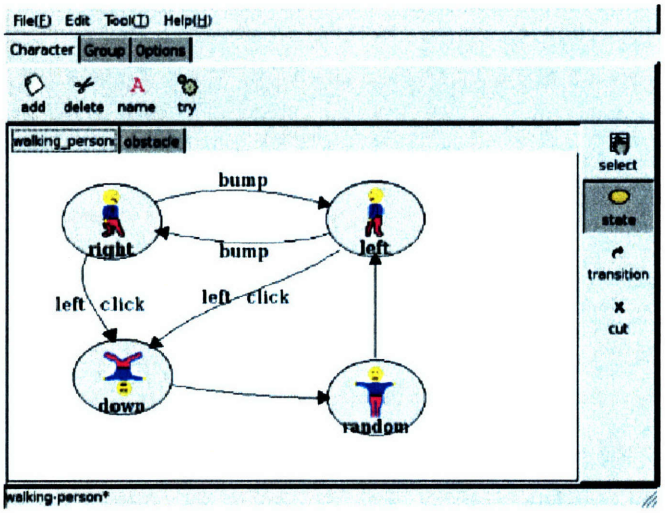


Figure 4: State-transition diagrams in Islay animation development environment

Multiple viewers may view and interact concurrently with a single completed Islay animation. For example, two viewers might play a game of soccer against one another; each player can view half of the screen and characters can move around the entire screen, thereby moving from one computer to the next. The view of this "couple play mode" is shown in Figure 5.

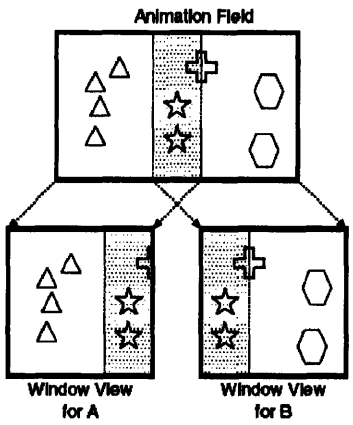


Figure 5: Couple play mode in Islay

Like NetMorphs and AgenTalk, entire objects are being transferred from one machine to another, which does not enable users to interact with the same objects at the same time. Additionally, Islay only enables users to interact collaboratively on *completed* projects; it does not provide an environment in which children can interact collaboratively while *constructing* projects.

2.4 MOOSE Crossing

In Amy Bruckman's Doctoral thesis (16), a text-based MUD (Multi-User Dungeon) called MOOSE (MUD Object-Oriented Scripting Environment) Crossing is presented. This environment enables children to create a virtual world together, programmatically adding their own places, objects, and creatures. In designing this shared world, the MOOSE Crossing community "provided support for learning through design and construction (16)."

NetScratch shares the goal of encouraging collaboration and support through a networked environment. However, NetScratch is different from MOOSE Crossing in a few ways. First, NetScratch has the advantage of a graphical user interface. NetScratch is also different from MOOSE Crossing in that users do not design projects by adding artifacts and inhabitants to a shared world, but rather NetScratch users each have their own world in which they can use real-time data from other people's projects, or "worlds," and from the Internet. This approach enables children to create a diverse set of personally meaningful projects, which can range from a personalized chat system to an interactive map portraying real-time weather data to a collaborative digital drawing board.

2.5 YAHOO! PIPES

Pipes (17) is an online service, provided by Yahoo!, that enables people to create their own mashups (a website or web application that combines information from multiple web sources) by mixing RSS feeds in different ways. Pipes are created in a visual editing environment, shown in Figure 6. In this environment, modules, which either fetch an RSS feed or manipulate the RSS feed data in some way, are connected by "pipes" that direct the output from one module to the input of another.

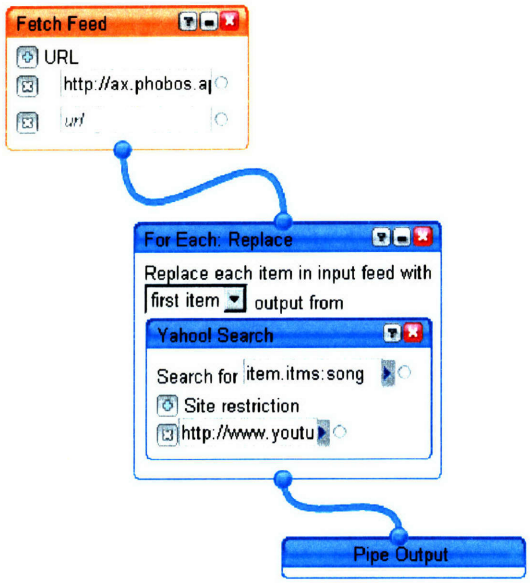


Figure 6: Yahoo! pipes enables users to create mashups by mixing and manipulating RSS feeds

Users of Yahoo! pipes can bring together data from different websites in new and creative ways. For example, an "Ultimate Sports Feed" pipe brings together sports updates from ESPN, Sports Illustrated, Fox Sports, and other popular sports news websites, into a single RSS feed. A "YouTunes" pipe gathers the top ten popular songs from the iTunes music website and locates the corresponding music videos on the YouTube video website.

Yahoo! pipes is similar to NetScratch in that they enable users to bring together information from the web in personally meaningful ways. However, the development environment and capabilities for pipes is quite different from NetScratch. While pipes provide more complicated ways to gather and manipulate information than NetScratch, they do not provide an environment in which users, especially

novices, can easily integrate the data retrieved into their own creative contexts. The output of a pipe is represented as an RSS feed, so only people who are comfortable with current web development tools would be enabled to present the data in a personalized and dynamic context.

2.6 SUMMARY

Both NetScratch and the related work presented in this chapter share the goals of enabling novice users to bring together networked information in personally meaningful ways and fostering a networked model of collaborative creating. However, these other systems are different from NetScratch in that they either do not emphasize a model of real-time dynamic user interaction, or do not offer a graphical programming environment in which children can use the networked information to create personally meaningful networked projects. Still, all of the environments presented above served as inspiration for the work done in this thesis.

3 CURRENT INTERACTIONS IN SCRATCH

Scratch programs are created by snapping together graphical command blocks. The basic layout of Scratch, as shown in Figure 7, includes four main areas:

- ***Blocks Palette***. List of programming blocks that can be used to build scripts.
- ***Scripts Area***. An area where blocks can be dragged in and snapped together to create scripts.
- ***Sprites List***. A list of all the sprites, or objects, in the current project.
- ***Stage***. A place where programmed sprites interact, and Scratch creations come to life.

Figure 7: Scratch layout

Scratch projects consist of multiple sprites that have a limited set of commands that represent their potential actions. These commands, available in the Blocks Palette, are divided into eight categories, based on how they affect the sprites; for example, a block in the Motion category controls how the sprite moves and a block in the Looks category controls how the sprite looks.

These commands are represented by graphical programming blocks that, as mentioned previously, snap together, much like LEGO bricks or jigsaw puzzle pieces. Like puzzle pieces, these blocks only snap together in ways that make sense. The blocks are then executed in sequence based on the order in which they are snapped together.

Many of the command blocks for sprites are self-contained in that they control only that sprite's specific actions. However, some blocks enable sprites to interact with other sprites, such as detecting when they are touching each other. Additionally, there are blocks that enable people to interact with their projects through keyboard and mouse inputs, as well as blocks that enable sprites to respond to data detected from the physical world using a special sensor board. All of these communications enable Scratch projects to be dynamic and interactive.

Along with blocks that communicate outside the realm of a single sprite, it is possible to export entire sprites from one project to another. This is another way in which projects can use information from outside its own realm.

The rest of this chapter walks through a sample project that utilizes these interacting functionalities, while Chapter 4 discusses how NetScratch expands on the interacting capabilities of Scratch, by enabling networked communications.

3.1 Blocks That Enable Interacting

3.1.1 Sprites Interacting with Other Sprites

Scratch projects contain multiple sprites that can interact with each other through command blocks. One way in which sprites can interact with each other is through command blocks that sense information about other sprites. Figure 8 shows a sample Scratch script, in which the *HungryBoy*[1] sprite points toward the *IceCream* sprite, and keeps moving until he detects that he is touching *IceCream*.

[1] The image used for *HungryBoy*, made by Doug McCarthy at Disney, was retrieved from the following website: http://cartoonmodern.blogsome.com/2006/03/15/inspired-by-cartoon-modern-part-2/ [Cited May 23, 2007]

HungryBoy's script | **Before script execution** | **After script execution**

Figure 8: *HungryBoy* sprite detects and moves toward the *IceCream* sprite

Another way in which sprites interact with each other is through global variables, created in Scratch by choosing a "For all sprites" variable option. An all-sprites variable can be changed or accessed by any sprite. In Figure 9, both *HungryBoy* and *IceCream* interact with an all-sprites variable called *percent left* that keeps track of how much ice cream is left. The *IceCream* sprite sets its size to the value of *percent left*, while *HungryBoy* becomes sad when *percent left* equals zero.

Sprite	Script	Result
IceCream	set size to percent left %	percent left 30
HungryBoy	if percent left = 0 switch to costume sad face	percent left 0

Figure 9: Two sprites within a single project share a variable called *percent left*

In addition to blocks that detect information about other sprites and variables that can be shared among sprites, there are also blocks that can send messages among sprites. To demonstrate this concept, *IceCreamTruck* is introduced into the sample project. *IceCreamTruck* uses a **broadcast <replenish ice**

cream> command block, which broadcasts a "replenish ice cream" message to the Scratch world. All other sprites in the project can listen for this message and trigger other commands upon receiving it. Figure 10 demonstrates an example of *HungryBoy* and *IceCream* reacting to the message sent by *IceCreamTruck*. A further discussion of the design of Scratch's broadcast mechanism is presented in David Feinberg's Masters Thesis (18).

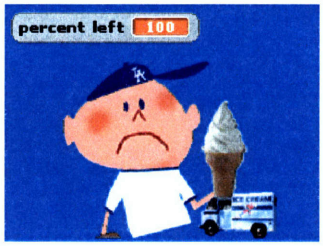
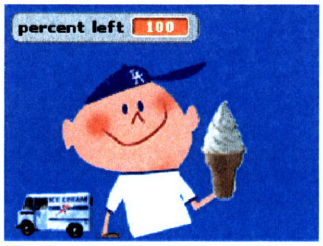
Sprite	Script	Result
IceCreamTruck	broadcast replenish ice cream	
IceCream	when I receive replenish ice cream set percent left to 100	
HungryBoy	when I receive replenish ice cream switch to costume happy face	

Figure 10: Broadcasting in Scratch enables sprites to send messages between each other

These interactions enable children to create projects that have interactive sprites; however, the sprites can only interact within a single project on one computer.

3.1.2 Users Interacting with Sprites

Scratch also has capabilities that enable *users* to interact with and control sprites in a Scratch project. Particularly, a Scratch project can detect and respond to keyboard and mouse inputs. To make the

previous sample project more interactive, *IceCream*'s position is dynamically set to the position of the user's mouse, using the code shown below in Figure 11.

IceCream's script | **Before script execution** | **After script execution**

Figure 11: User interacting with a sprite through mouse inputs

3.1.3 Sprites Interacting With the Physical World

In addition to interacting with mouse and keyboard input, sprites can interact with the physical world by connecting a sensor board to Scratch (19). In Figure 12, the value of a light sensor simulates the sun melting the ice cream.

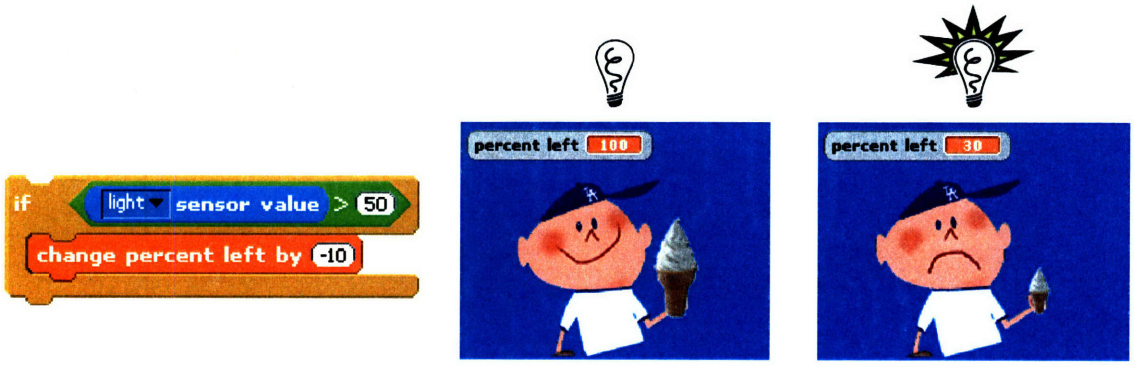


Figure 12: Sprite interacting with the light detected from the physical world

These capabilities allow children to interact with the projects they create on the screen in a physical way. However, this interaction is still limited to a single computer in a single location.

3.2 Sharing Static Sprites and Project Ideas

In addition to blocks that enable interaction between sprites in a project and with the physical world, there are also ways to incorporate components of one project into another. This sharing can be divided into two main categories: 1) sharing project ideas and 2) sharing sprites.

3.2.1 Sharing Project Ideas

The Scratch website (2) is a place where the community of Scratch users can upload their projects, view other users' projects, create galleries, and communicate through a forum. Figure 13 shows a screenshot of the Scratch website homepage, which displays the "Boy and Ice Cream" sample project created in this chapter.

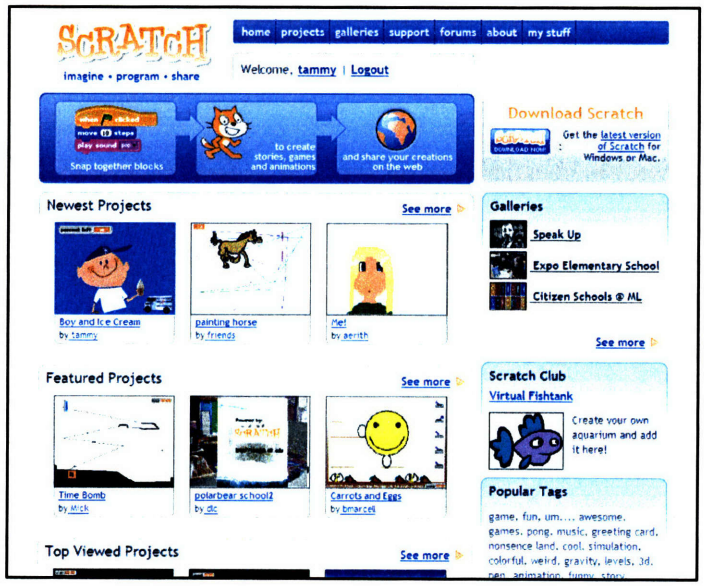


Figure 13: Scratch website for sharing projects and ideas

The Scratch website encourages collaboration among Scratch users and provides a space where children can share their ideas. One way in which people get ideas for projects is by finding a stimulating project, downloading it, and modifying parts of it. For example, someone might see "Boy and Ice Cream," and become inspired to create a derivative "Girl and Ice Cream" project.

Once the project is downloaded from the Scratch website, it is limited to interactions on the local machine. Thus, the Scratch website only provides a way for children to interact and collaborate on their projects asynchronously; it does not enable children to share their projects, or parts of projects, dynamically in real-time.

3.2.2 Sharing Sprites

In addition to sharing project ideas on the website, there are ways to incorporate particular components from one project into another. A sprite can be exported with all of its costumes, sounds, and scripts, and then imported into another project. For example, the *HungryBoy* sprite created in this chapter could be exported from its original project and then imported into a new project. However, the two *HungryBoy* sprites are not linked in any way. That is, changing the costume of *HungryBoy* in the original "Boy and Ice Cream" project will not affect the costume of *HungryBoy* in the new project.

3.3 Pros and Cons of Current Means for Interactions

The currents means for sharing and connecting within Scratch emphasizes a model of self-contained projects. This model is beneficial because projects can be run anywhere without dependence on Internet connection or other users.

However, there are two main limitations of the current Scratch model. First, Scratch projects are limited to the information that is available on one's local machine. Second, the current model supports only static sharing. Once an idea, a project, or a sprite has been shared, there is no way to dynamically update or collaboratively interact with those artifacts. It becomes a sequential, individual, back-and-forth process of collaboration. The following chapter introduces NetScratch, which addresses these two limitations.

4 INTRODUCING NETSCRATCH

NetScratch is an extension to Scratch that integrates networking capabilities. These capabilities enable children to create projects that depend on information from outside of their own projects – particularly, real-time information from the web and other projects. This chapter discusses the principles that drove the design of NetScratch and the final NetScratch additions.

4.1 DESIGN PRINCIPLES

The goal of this thesis is to make Scratch network-aware in a way that is conceptually straightforward, easily-used, and evocative. The design of NetScratch takes into consideration how these features integrate into the already-established Scratch user interface, how children interact with these features, and how these features best enable children to be creative and collaborative. Specifically, the design principles that are most important are:

- ***Tinkerability***. Children should feel able to "tinker" with the new functionality by quickly and easily interacting with it and observing the outcome.
- ***Consistency***. The resulting features should make sense within the current Scratch feel and fit with other Scratch metaphors.
- ***Collaborability***. The new functionality should provide an easy way for children to work on projects together.
- ***Transferability of important ideas***. The features should provide children with insight into how distributed web technologies are designed and created. Particularly, it should expose ideas about computation, networks, web mashups, and real-time data.
- ***Low floor***. As suggested by Resnick and Silverman, NetScratch should contain "features that are specific enough so that kids can quickly understand how to use them (low floor), but general enough so that kids can continue to find new ways to use them (wide walls) (12)." Specifically, because Scratch is meant to be used in informal settings, the functionality should be straightforward and easily understood, avoiding the use of jargon or other complex notions.
- ***Wide walls***. As mentioned in the previous design principle, the NetScratch features should be basic enough that children can use them in personally meaningful ways, providing new

opportunities for creative expression. Particularly, the features should appeal to children with different interests and personalities, enabling them to each create their own variety of projects.

4.2 Choosing the Building Blocks

The purpose of NetScratch is not for children to think about the network in connecting to websites, accessing networked information, or connecting to other people's projects. Rather, children should feel empowered to integrate the networked information seamlessly into their project ideas.

Resnick and Silverman advise, "the choice of the basic 'building blocks' in a programming language determines what kids are likely to learn as they use the language (12)." Taking their advice into consideration and the goal of encouraging children to use networked information creatively, I chose to de-emphasize underlying network communications in NetScratch. For example, the children would not be exposed to the data retrieval and parsing needed to incorporate networked data into their projects. Instead, the blocks presented to the kids would simply report information from the web or from other people's projects, emphasizing the creative uses of this information, without exposing the underlying implementation details.

4.3 Initial Ideas

There were a couple of possibilities for how networked information might be presented to a NetScratch user. Each way had design trade-offs, as discussed below, and the ultimate design was chosen to best align with the design principles discussed above.

4.3.1 Two Kinds of Variables?

Scratch currently contains a "Variables" category, which enables children to create and manipulate variables in their projects (See Section 3.1.1). One option for integrating networked information into NetScratch was to combine the ideas of incorporating information from the web into a project and information from other projects into a project. Both types of information are variables that can change in real-time through the life of a project. The main difference is that information from the web (e.g., the

temperature in Cambridge, Massachusetts) usually cannot be changed by a friend or by the person using the information, while information shared between projects can be shared by the users themselves. In other words, information from the web is like a read-only variable and information from other people's projects is like a read/write variable. Thus, the idea was to integrate web variables and other people's variables into the "Variables" category, allowing some variables to be manipulated (read/write) and others just fed into the project (read-only).

Although one could argue that it is important for children to understand the parallels between web variables and other people's variables, combining these ideas into one notion seemed to be more confusing than educational. Ultimately, it is not the main goal of NetScratch to enable children to understand different types of variables. Instead, it is important for them to feel empowered to use this networked information in their project in new and creative ways. Taking that into consideration, I decided to separate these ideas of 1) sharing information between projects and 2) using information from the web. Thus, I proposed the following alternative.

4.3.2 Variables and Sensors

Scratch currently contains a "Sensors" category with blocks that sense various kinds of information within a project (see Section 3.1). An example of information that can be sensed from within a project includes whether a sprite is currently touching another sprite. Information that can be sensed from a user includes keyboard and mouse input. And information, such as light level and slider position, can be sensed from the physical world using an augmented sensor board (19).

These sensors can be detected, or read, but not changed, or written to. Therefore, a logical extension to this paradigm was to categorize information detected from the web as "web sensors." And, as suggested in the previous section, information shared between projects would be represented as variables, which could be both read from and written to. These read/write variables fit into the existing variable paradigm.

4.4 The Final Design

In the final design, NetScratch's networking capabilities are split into two categories. *Websensors* are an additional type of sensor block and *shariables* are a new kind of variable. Additionally, to offer more flexibility to the types of information that can be shared, NetScratch introduces "Text," or strings, as a new data type.

4.4.1 Adding String Variables

To handle richer forms of data, I added a string data type to Scratch, in addition to the number and Boolean types it already had. Data types in Scratch are represented by the shape of the reporter block; numbers have rounded edges, Booleans have pointed edges, and strings were added with square edges (see Figure 14).

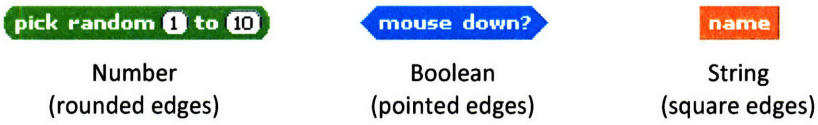


Figure 14: Data types in Scratch are represented by their shapes

Strings were introduced into Scratch with simple getter and setter blocks as shown in Figure 15. Eventually, strings will be more integrated into the Scratch experience, providing more string manipulation and usage. A discussion about extending string functionality is presented in Section 8.1.3.

Figure 15: New string variable blocks in NetScratch

4.4.2 Adding *Websensors*

Table 1 presents the set of *websensors* that were incorporated into NetScratch, along with a sample output of the corresponding data sensed from the web. These blocks are all reporters that return either

integers or text. For example, the **temperature in <02139>** block returns a number representing the current temperature in Cambridge, Massachusetts, the **another word for <fun>** block returns a text synonym for the input parameter, "fun," and the **Scratch project of <tammy>** block returns the name of a random Scratch project submitted to the Scratch website by user *tammy*.

Block	Example output
temperature in 02139	temperature in 02139 38
another word for fun	another word for fun pleasant
word of the day	word of the day proximity
joke of the day	joke of the day What do you break by making noise?
random question	random question Do more people prefer dogs or cats?
Scratch project of tammy	Scratch project of tammy TurtleArt

Table 1: *Websensor* blocks and example outputs

Some of the blocks have drop down menus that offer different types of information based on a certain parameter. For example, for a particular zip code, one could retrieve various types of data for that area including temperature, weather conditions, humidity, visibility, wind chill, wind speed, wind direction, latitude, longitude, sunrise, sunset, city, state, and time (see Figure 16).

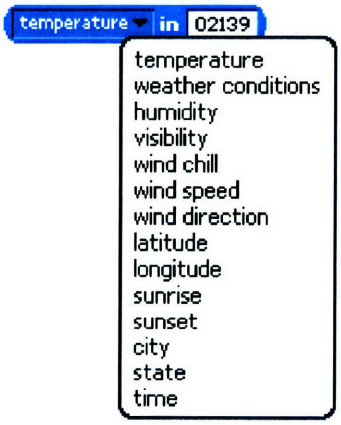


Figure 16: Temperature *websensor* options

To integrate *websensors* programmatically, *websensor* blocks can be dragged into command blocks. Specifically, a *websensor* that reports a string value can be dragged into a **say <>** block, a **think <>** block, or an **<> equals <>** block. If the *websensor* reports a number value, it can be dragged into any block that has a number as a parameter, such as a **move <> steps** block or a **set image effect to <>** image manipulation block.

Figure 17 presents an example of *websensor* scripts in the context of the "Boy and Ice Cream" project from Chapter 3. Now, instead of the ice cream melting based on the light detected in the room, the melting is controlled by the actual temperature in Cambridge, Massachusetts.

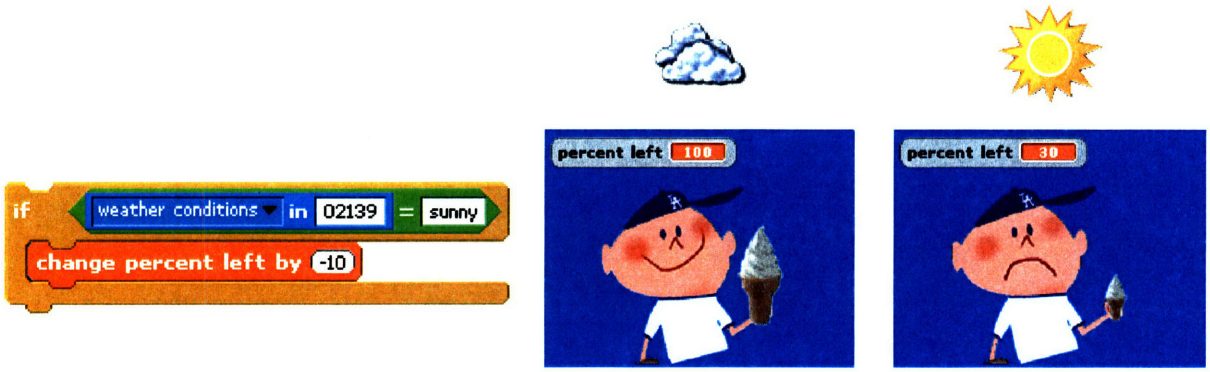


Figure 17: Sprites interacting with the actual temperature in Cambridge, Massachusetts

Additionally, this project can be extended to make *HungryBoy* voice his feelings about his ice cream in an interesting and dynamic way. Figure 18 presents a script that programs *HungryBoy* to say a synonym for "sad" when there is no ice cream left and a synonym for "happy" otherwise.

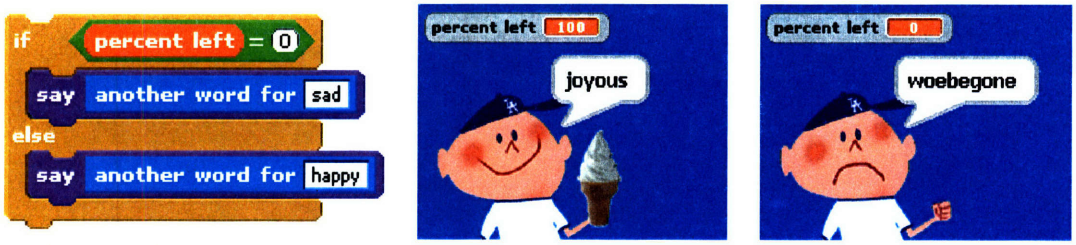


Figure 18: A sprite says dynamically-changing synonyms from a web thesaurus

The blocks presented in this section were available in NetScratch for the user study. However, they are meant to be only representative of the types of data that could be included from the web. Ultimately, these *websensors* could be extended by adding new ones, as well as including user-generated *websensors*, as discussed in Section 8.1.1.

4.4.3 ADDING *SHARIABLES*

In the final version of NetScratch, *shariables*, or shared variables, are presented as an option in the new variable creation dialog box, as shown in Figure 19. Building on the paradigm of variables available to only individual sprites (For this sprite only) and variables available to all sprites in a project (For all sprites), *shariables* (Shared) are available to *all* sprites in *any* project. Additionally, string (Text) variables, as mentioned previously, were added as an option.

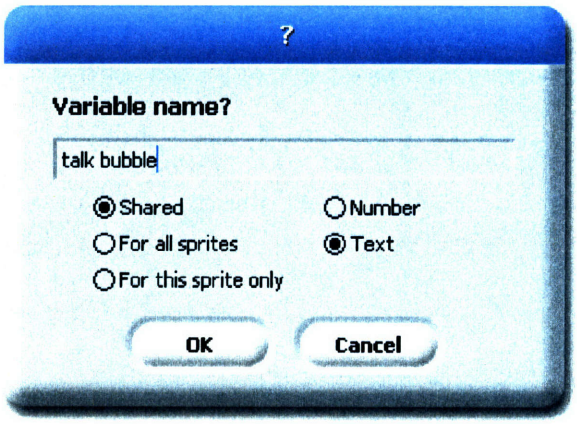


Figure 19: NetScratch variable creation dialog box

Once a *shariable* is created, it is added to a central server of *shariables*, which is available to anyone using NetScratch from any computer. The server also provides persistence for *shariables*; the values are stored even when a project is closed. To use an existing *shariable*, there is an "Import a variable" button in the "Variables" category, as shown in Figure 20, which lists all *shariables* on the server and their corresponding values.

Figure 20: Importing a *shariable* in the NetScratch user interface

Once a *shariable* is imported into a project, the value can be used in scripts, as well as changed by using the **set shariable to <>** block. Once the value is changed, it will update on the server, and any other project using that *shariable* will receive the latest value.

Using *shariables* broadens the scope of the "Boy and Ice Cream" sample project. Now, *HungryBoy* says the value of the *talk bubble shariable*, which can be set from any computer.

Figure 21: A *shariable* set on one computer affects a project on another computer

The project can be further extended by enabling it to respond to interactions occurring on another NetScratch project. For example, in Figure 22, another project running on Computer 1 features a game

in which an ice cream truck visits different neighborhoods delivering ice cream to children. One of the neighborhoods in that game is called *HungryBoy's neighborhood*. Whenever the ice cream truck visits *HungryBoy's neighborhood*, a *shariable* is changed on Computer 1, which triggers the arrival of the *IceCreamTruck* sprite on Computer 2. Figure 22 displays the code used to program these interactions and the corresponding project behavior.

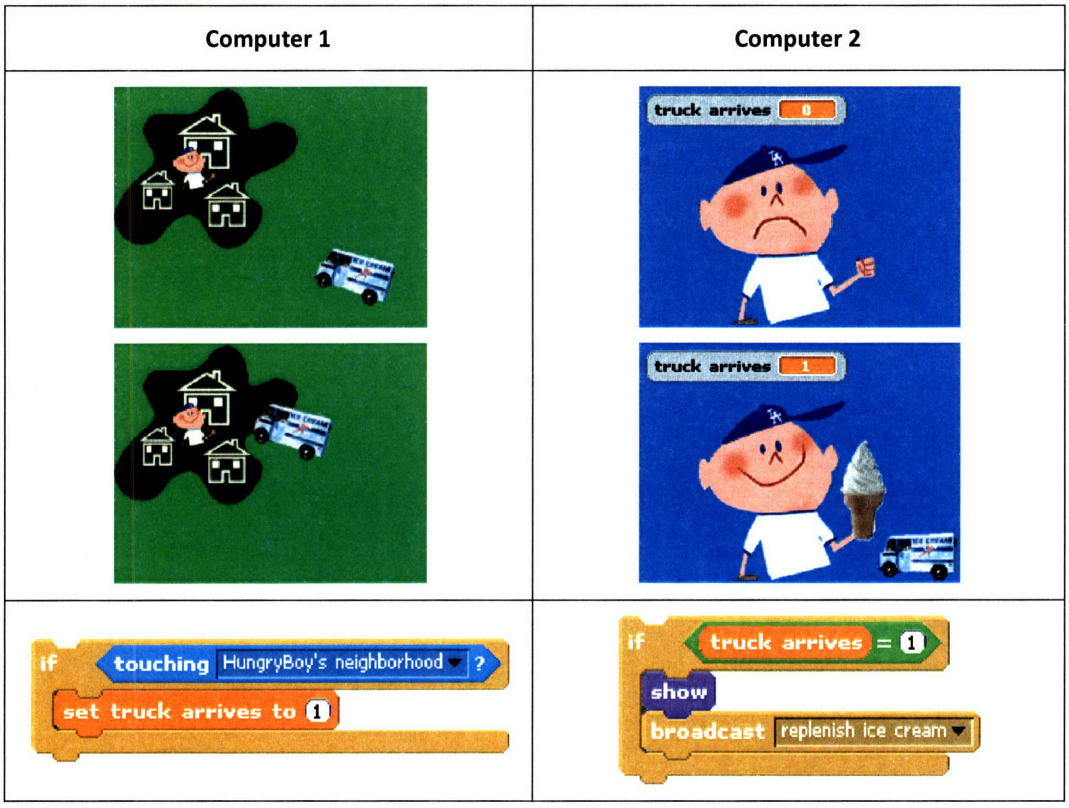


Figure 22: Changing the value of a *shariable* in one project triggers the arrival of *IceCreamTruck* sprite in another

These *shariables* enable children to create projects that have shared components, providing a platform for creating projects that dynamically interact in real-time.

5 NETSCRATCH USAGE SCENARIOS

NetScratch is meant to be an evocative technology that suggests a wide range of creative possibilities. With a better understanding of how one might interact with NetScratch, it will become easier to determine possible future directions for it. This chapter together with Sections 7.4.1 and 7.4.2 present an overview of the range of projects that can be created using NetScratch.

5.1 *WEBSENSOR* SAMPLE PROJECTS

5.1.1 JOKING SPRITES

The project presented in Figure 23 is a simple animation with one character telling a joke to the other one. Because the joke is pulled from a website[2] that changes every day, the project will have different behavior each day that it is executed.

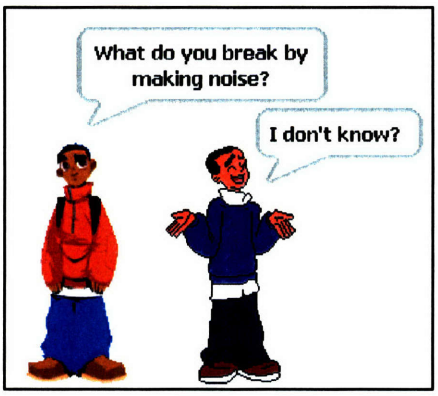


Figure 23: Simple *websensor* sample project with a joking sprite

5.1.2 MADLIBS LETTER

The project presented in Figure 24 uses *websensors* in a way that leverages real-time randomly changing data. It is a MadLibs letter that displays synonyms, generated by the **another word for <word>** *websensor* block, for the following words in the letter: "Dear LLK, You are the best. I am inspired by

[2] This *websensor* pulls information from a Yahoo! site that provides kid-friendly jokes: http://kids.yahoo.com/jokes/index [Cited May 23, 2007]

each of you. Thanks for everything. Sincerely, Tammy." The synonyms are represented in the blue boxes and periodically update to display different versions of the letter.

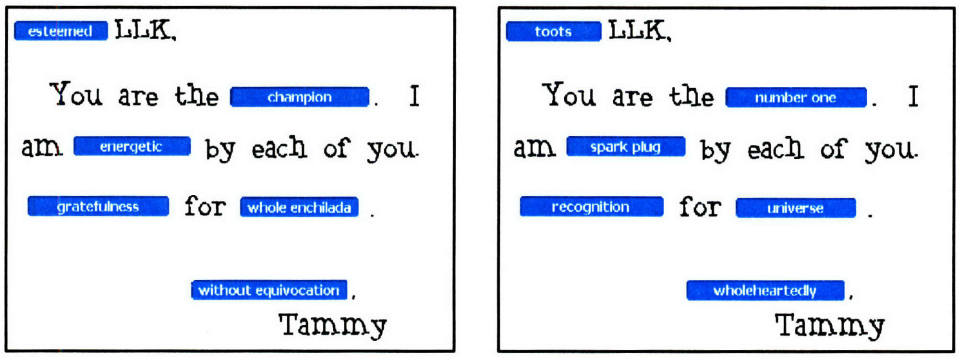


Figure 24: Dynamically-changing MadLibs Letter

5.1.3 GADGET: INTERACTIVE MAP OF USA

Gadgets, sometimes known as widgets, are "miniature objects ... that offer cool and dynamic content that can be placed on any page on the web (20)." Gadgets have become more popular in the context of Google gadgets, Mac's dashboard widgets, and Yahoo!'s widget collections. The NetScratch gadget presented in this section is a randomly-generated graphical and interactive representation of a map of the United States of America. Each time the project is restarted, 50 new random zip codes are generated and a dot is moved to the corresponding latitude and longitude location for each zip code. The color of the dot depends on the current temperature in the location. Figure 25 shows two randomly-generated maps. One can see at a glance that the northeast is cooler than the south.

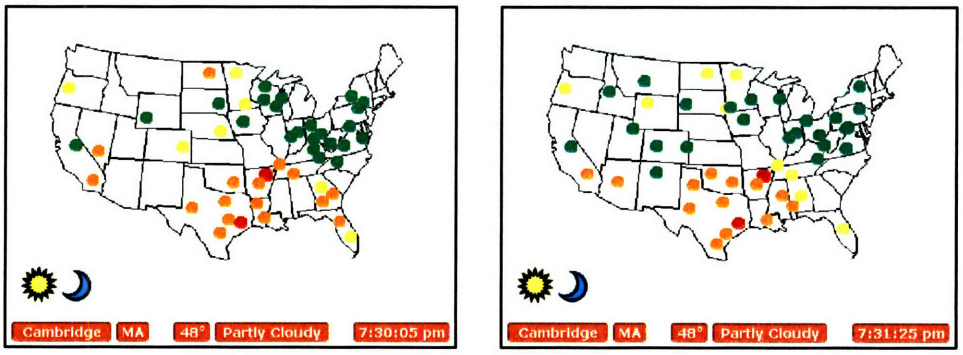


Figure 25: Two randomly-generated maps of the United States of America

As shown in Figure 26, clicking on a dot populates the text fields on the bottom with the corresponding city, state, temperature, weather conditions, and time.

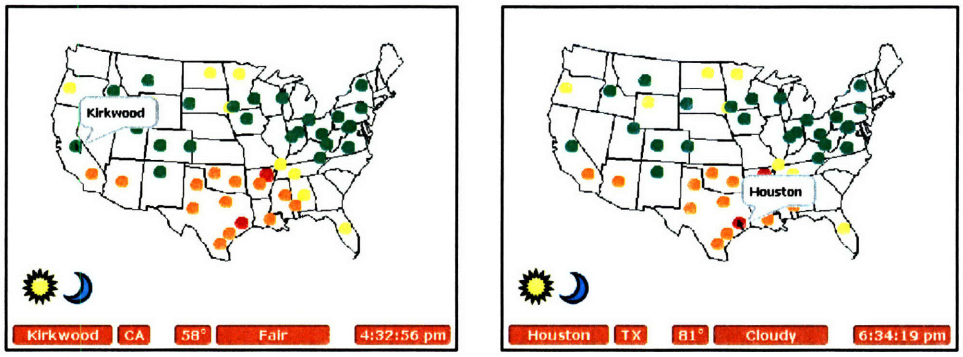


Figure 26: Clicking on a dot dynamically displays information associated with that location

Clicking on the sun or moon broadcasts a message to each dot, triggering it to present its corresponding sunrise or sunset time. The result of clicking on the moon is presented in Figure 27.

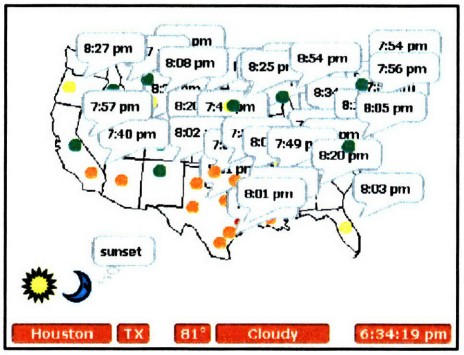
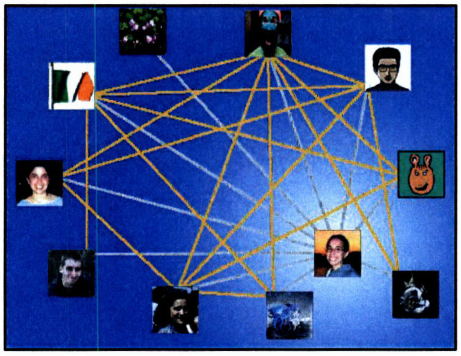


Figure 27: Clicking on the moon displays the sunset time for each location

5.1.4 WEB OF FRIENDS

Online communities such as MySpace and Facebook are becoming increasingly popular among children. These communities serve as a space for social interactions, and members often maintain a list of the people with which they socialize, also known as their "friends." The NetScratch project in Figure 28 (a) presents a visualization of the connectedness of a group of friends from the Scratch website. A line between two people indicates that they are friends with each other on the Scratch website.

This project uses the **Scratch friend of <Scratch username>** *websensor* block. In particular, a line is drawn, using the Scratch pen command, between two sprites *Aardvark* and *jay* if the result of **Scratch friend of <Aardvark>** , which returns a random Scratch friend of *Aardvark's*, ever returns *jay*. The script for detecting whether *jay* and *Aardvark* should be connected in this project is shown in Figure 28 (b).



(a) output of "Web of Friends" project

(b) script detecting if two users are friends on the Scratch website

Figure 28: "Web of Friends" project detects information from the Scratch website

When I created this project, I specifically chose friends from different locations to see if any of them were connected; particularly, the project includes friends from various parts of Massachusetts, Texas, Ireland, France, and Hungary. To my surprise, this project showed me that many of my friends from all different locations are friends with each other as well. By running this project again over time or with different friends, I can visualize my friend networks in a variety of ways.

5.2 *SHARIABLE* SAMPLE PROJECTS

5.2.1 CUSTOMIZED CHAT

Chat is quite popular among the current generation. However, most people use pre-defined chat systems that have been designed and developed by other people. The sample project in Figure 29 presents a customized chat system in which NetScratch users can create their own avatar and make words appear in talk bubbles above their own picture. This project could be extended to enable the users to dynamically change their positions, pictures, and background images. In Section 7.4.2, an

example of the beginnings of a customized chat system, as designed by 13-year-olds during the user study, is presented.

Figure 29: Customized chat system created using *shariables*

5.2.2 MOOD-CHANGING PROJECT

NetScratch gives users the ability to connect with one another in new and original ways. Instead of simply enabling chat, the project shown in Figure 30 presents a group of NetScratch users and their current mood. Each user can change their mood from their own computer, and it will be reflected in this project. By keeping this project running throughout the day, NetScratch users can easily see how their friends are feeling, and quickly and unobtrusively alert the group of their current feelings.

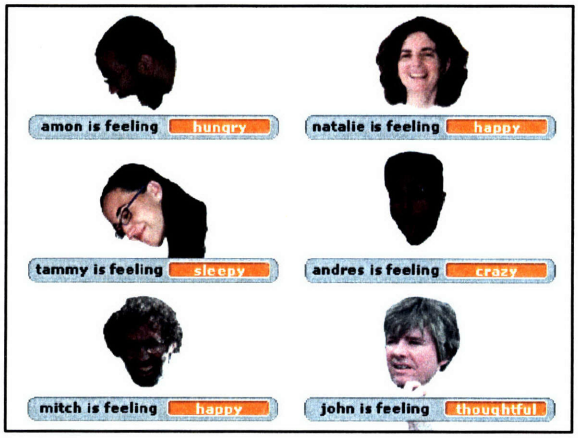


Figure 30: Project where friends can update their moods

5.2.3 MULTIPLAYER GAMES

Online multiplayer games are becoming increasing popular among children nowadays. Using NetScratch, children can develop their own customized online games. The project presented below shows a two-player pong game in which each player controls one paddle.

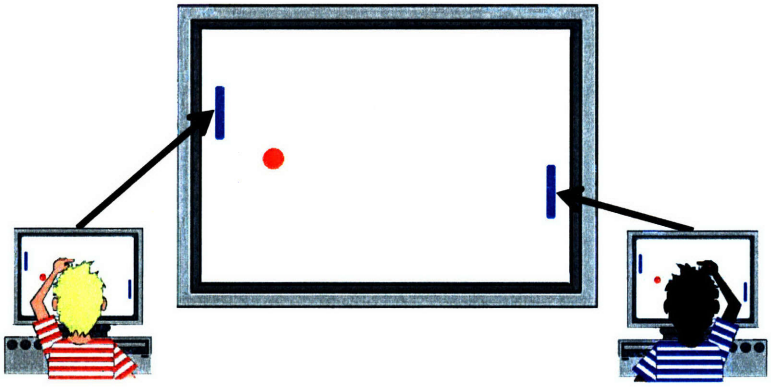


Figure 31: Two-player networked pong game using *shariables*

The position of each paddle is set by the arrow keys on the local machine. Additionally, a *shariable* is set to the position of each paddle, thereby enabling a networked computer to read the value of the *shariable* and dynamically update the position of the remote paddle.

5.2.4 HIGH SCORE LIST

Generating a persistent high score list has been a commonly-requested feature among Scratch users. The NetScratch project in Figure 32 presents an example of a persistent high score list that can be dynamically updated by Pong players on many different computers.

This high score list is maintained by keeping a group of *shariables* (named, for example, *first place player*, *first place score*, *second place player*, *second place score*, etc.) to remember the top five high-scoring Pong players. If a new player achieves a score that should go on the high score list, then that player and score get stored in the corresponding *shariables* based on their score place, pushing all the players with lower scores to be stored in the successive *shariables*, thus maintaining a persistent and shared high score list.

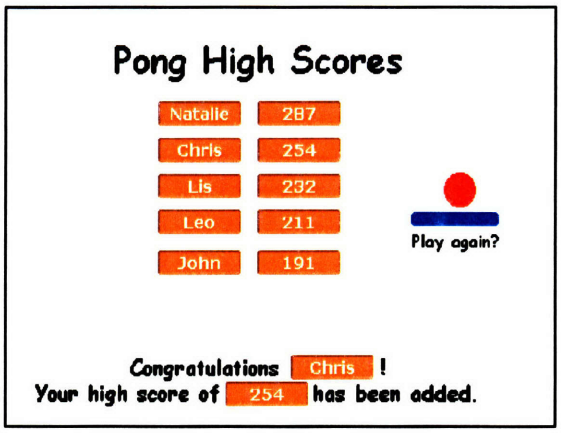


Figure 32: A game with a persistent networked high score list

5.2.5 HOOK-UP *SHARIABLES*

Hook-ups (19) are electronic sensor boards that can be attached to Scratch to enable children to interact with their Scratch projects through physical interfaces. The project presented in this section extends this notion by enabling children to interact with *other people's* Scratch projects through physical interfaces. Specifically, the project, shown in Figure 33, enables two friends to synchronize the times that they wake up in the morning and go to sleep at night. Whenever one friend turns the lights on, the rooster crows and wakes the other. Similarly, whenever one turns the lights off, a lullaby is triggered, reminding the other to go to sleep.

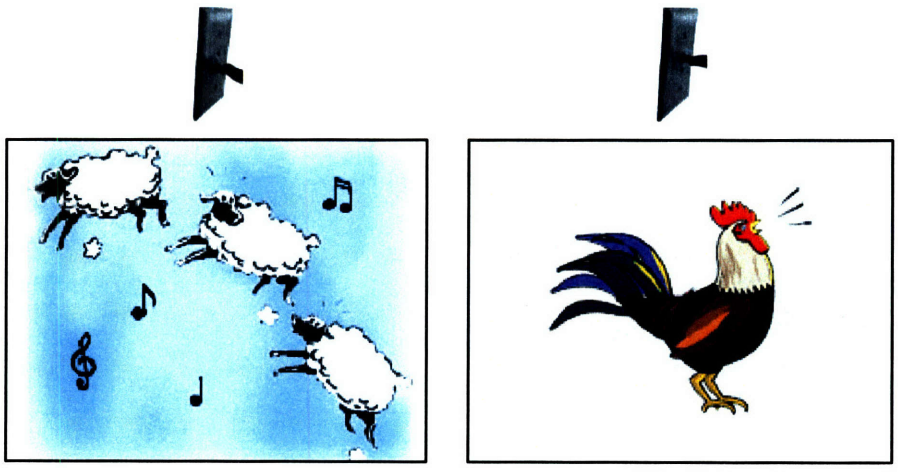


Figure 33: NetScratch project as a physical communication device

To create this project, each friend has a *shariable* that changes value whenever the lights in her room go above or below a certain threshold. For the purposes of this example, the friends' names are Emily and Karina. Emily creates a project with the following two scripts, which reset the value of the *emily's light shariable* based on the light detected in her room.

Figure 34: Scripts that reset the value of a *shariable* depending on whether the lights are on or off

While the above scripts are being run in Emily's project, Karina's project runs the following two scripts, which either display the rooster or sheep scene, depending on the value of *emily's light*.

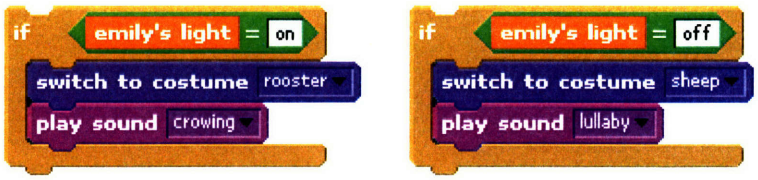


Figure 35: Scripts that display a different scene based on the value of a *shariable*

These scripts can be duplicated, replacing *emily's light* with *karina's light*, enabling the friends to communicate with each other in a personally meaningful way.

5.2.6 SCRATCH-A-GOTCHI

Tamagotchis (21), popular in the 1990's, are handheld digital pets that require feeding, playing, and cleaning in order to keep them healthy. More recently, digital pets have become prevalent on the web through websites such as Webkinz (22). On this website, users can take care of Webkinz pets in a networked community.

The sample project[3] in Figure 36 shows a Scratch-a-gotchi pet that NetScratch users can create, deciding what it should look like, how it should behave, and how to keep it healthy. Then, the community can get together to care for this dog, by petting it often and feeding it just the right amount.

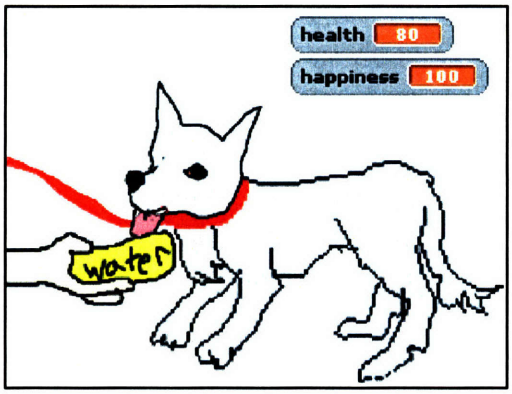
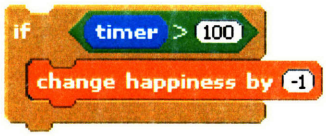


Figure 36: Scratch-a-gotchi

This project uses two *shariables* that keep track of the dog's health and happiness. Whenever the dog is touched, or petted, the *happiness shariable* increases by 1. However, if a lot of time goes by and the dog doesn't get any attention, the *happiness shariable* slowly decreases.

Whenever the dog is pet, it gets happier



If the dog goes too long without any attention, its happiness will decrease

Figure 37: Scripts that control the happiness of the Scratch-a-gotchi

Because the pet is created in the NetScratch programming environment, other users can implement their own additions that control the dog's behavior – perhaps a way to take it out for a walk or have it make friends with other dogs.

[3] This project was originally a Scratch project, developed by a Scratch user with username *mina*. The original project is available here: http://scratch.mit.edu/projects/mina/962 [Cited May 17, 2007]

5.2.7 COLLABORATIVE PAINT

Paint tools on computers, such as KidPix, Photoshop, and Microsoft Paint, are popular among children, enabling them to invent their own digital artistic creations. This sample project, presented in Figure 38, extends the common single-user painting experience to a collaborative dynamic experience in which multiple people from different computers can contribute to a painting.

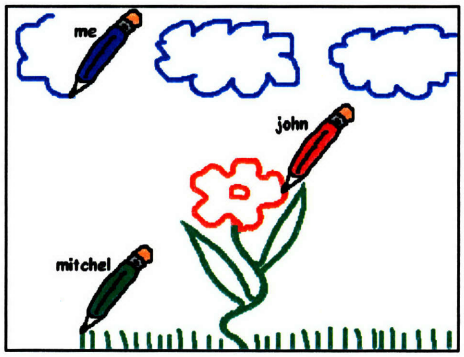


Figure 38: Collaborative networked digital drawing board

Each painter in this project has a set of *shariables* which update based on the position of the pencil on his or her local machine. Then, the other machines that are running this project automatically change the position of the remote pencil based on the vales of the *shariables*, leaving a pen trail as the pencil moves. The real-time updates of the pencil positions thereby enable multiple painters to collaborate dynamically.

6 IMPLEMENTATION

NetScratch is implemented as an extension to the Scratch programming environment, which is built in the Squeak programming language. This chapter presents an overview of the interesting parts of the implementation, offering insight into how the current NetScratch framework is used to support *websensor* creation and *shariable* maintenance. The chapter also offers recommendations for how to improve the current implementation of NetScratch to handle the scaling issues that would arise as a result of an increased number of NetScratch users.

6.1 *WEBSENSOR* IMPLEMENTATION

The implementation of *websensors* includes a method to grab HTML documents from the web and a method to parse that information to extract the desired *websensor* data.

6.1.1 ACCESSING INFORMATION FROM THE WEB

To access information from the web, a fetcher method built into Squeak retrieves HTML from a specified server and pathname. An *ActiveFetchers* list keeps track of all of the relevant websites that need to be polled to retrieve the desired *websensor* data. Multiple times each second, NetScratch pings all of the websites on the *ActiveFetchers* list.

When implementing the website accessing code, the first obstacle I encountered was the time delay for retrieving HTML documents. Because it takes a variable amount of time, depending on factors such as network bandwidth and congestion, scripts that depend on the value of a *websensor* would hang while waiting for a response. To partially address this problem, the *ActiveFetchers* list is pre-seeded with websites based on a set of default parameters. That way, NetScratch has a local copy of potentially desirable *websensor* data that can be immediately returned if a script accesses it.

However, because some *websensor* blocks allow the user to type arbitrary strings as parameters, such as the **another word for <word>** block shown in Figure 39, *ActiveFetchers* cannot be pre-seeded with all possible parameter values.

Figure 39: Arbitrary values can be typed in as parameters to *websensor* blocks

Because of these unpredictable parameters, there was still the problem of whether or not a script that depends on not-yet-retrieved *websensor* data should hang until the value is available, or return a temporary "Waiting.." value. Both of these options had downsides: hanging would delay a script for an arbitrary amount of time, and returning a temporary value would forfeit the actual response a user is expecting for an immediate placeholder response.

Because hanging for an arbitrary amount of time would result in unreliable efficiency of NetScratch programs, a version of the immediate-temporary-value solution was implemented. Specifically, a *websensor* returns a temporary "Waiting.." value the first time a user accesses it; after it is accessed the first time, it is added to the *ActiveFetchers* list, continuously updating for the rest of the NetScratch session, and caching the latest value in case the user accesses it again.

To further optimize the availability of *websensor* data, NetScratch always extracts all of the potentially desired *websensor* data from a particular website. For example, even if only the temperature for a particular zip code is requested, all the information associated with that webpage is also parsed, including the temperature, weather conditions, wind speed, and so on. That way, NetScratch is ready with other information that the user might request in the future.

While the above approaches addressed the problem of users expecting the *websensor* results immediately, it introduced problems of pinging the same websites over and over again. Some websites interpreted these pings as Denial of Service attacks and reacted by denying further requests for several minutes. As a temporary solution to this problem, the frequency of pings to these websites was reduced.

The next version of NetScratch would benefit from having a dedicated server, acting as a cache, which would handle repeated accesses to relevant websites. The dedicated server could even cache the entire databases of relevant websites, avoiding repeated pinging based on specific parameters. This approach would solve the problem of perceived Denial of Service attacks, since the server could time the website

pings more appropriately outside of the confines of the Scratch environment, or eliminate pinging completely. Assuming the dedicated server has a fast response time, it would also address the problem of needing to return temporary "Waiting.." values upon the first *websensor* access because the server could store a greater amount of data from websites.

6.1.2 Parsing HTML Documents

Once documents are retrieved from the web, NetScratch parses the HTML to retrieve the desired information. Developing a framework to systematically parse HTML documents would provide capabilities for the following extensions to NetScratch:

- Developers could easily add new *websensors* from within the NetScratch code
- Expert users could create their own *websensors*, by providing the required parameters
- Children could create their own *websensors* if they understood the parsing framework
- A program could be developed to create *websensors* on the fly using a browser add-on

Initially, it was not obvious how to create a standardized algorithm that could consistently parse a variety of websites. However, after implementing a few *websensor* parsers, an elegant solution emerged, in which these ad-hoc algorithms could be reduced to the 4-parameter scheme presented below.

The framework to parse the web pages is based on a pattern match using four parameters: 1) a unique pre-string, 2) an immediate pre-string, 3) an immediate post-string, and 4) a unique post-string. The algorithm basically sandwiches the desired data between the pre-strings and post-strings. Specifically, the algorithm first eliminates all the information before the unique pre-string and after the unique post-string, leaving only the information that is sandwiched in between. A similar process is then done using the immediate pre-string and the immediate post-string, leaving only the desired information in between. While this pattern-matching technique is still ad-hoc and relies on human judgment to select parameters that uniquely identify the desired field, it was a pleasant surprise that all 23 *websensors* that were implemented could use the same 4-parameter matching framework. Figure 40 demonstrates the parsing process for retrieving the temperature.

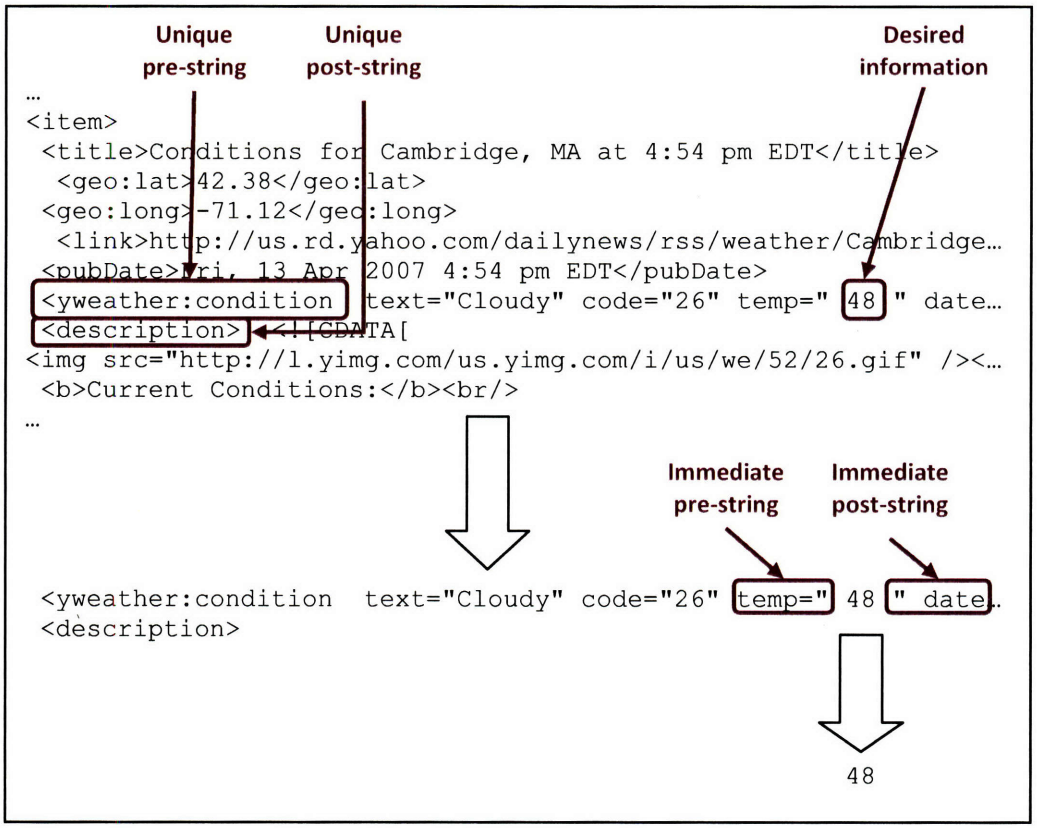


Figure 40: Four-parameter pattern-matching algorithm used to extract information from HTML documents

This parsing code was developed on the client side of the NetScratch application. However, because the HTML structure of a webpage may not remain constant over time, a better model would have the *websensor* parsing code on the server, where it can be updated centrally on the server, as opposed to requiring users to update their NetScratch applications. Additionally, new *websensors* could then be added to the server, which would then be available to all NetScratch users.

6.2 *SHARIABLE* IMPLEMENTATION

The *shariables* infrastructure is based on a client-server model. The server maintains the values of all the *shariable* data and the NetScratch client application periodically pings the server to 1) update the server values with the changed values from the local NetScratch application and 2) retrieve the updated values of the *shariables* that were changed remotely. More specifically, the communication between the server and client is as follows. The client maintains a mapping, *ClientMapping*, of *shariable* names to

values. *ClientMapping* contains only the *shariables* that are currently imported into that client's local NetScratch application. Each mapping has a flag denoting whether or not the *shariable* was changed locally. The client regularly sends *ClientMapping* to the server, via communication message */updatevars/<ClientMapping>*. The server contains *ServerMapping*, a mapping of all *shariables* existing on the network. The server updates its own mappings with the values which were flagged as changed in *ClientMapping*. Then it updates the remaining *shariables* in *ClientMapping* with the most recent values *ServerMapping* and returns the updated *ClientMapping* to the client NetScratch application. Figure 41 illustrates this process.

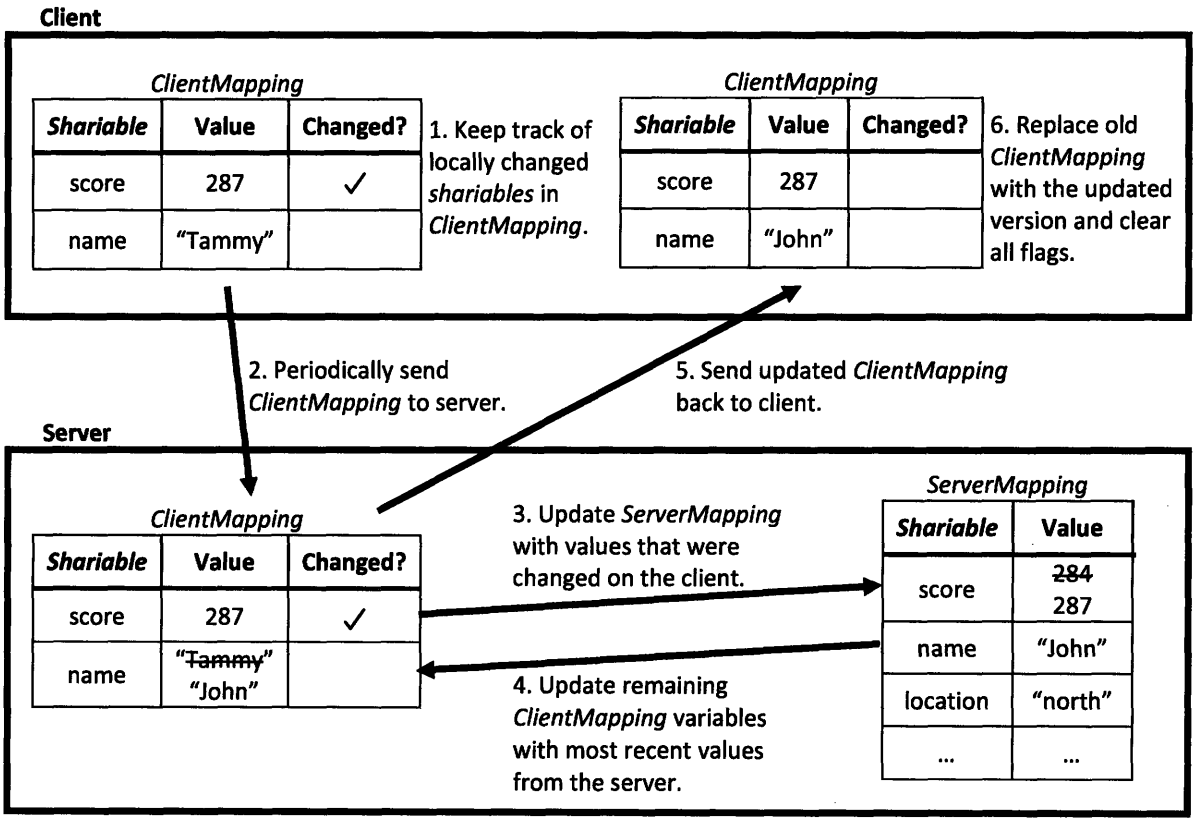


Figure 41: Client-server model for maintaining *shariables*

This simple interaction model, composed of sending the entire *ClientMapping* back and forth between the client and server, is easily portable to a dedicated server, which would better handle scale. In a more sophisticated model, I would recommend adding metadata to each *shariable*, including when it was last updated, who last updated it, and who has read/write permissions to the *shariable*. Also,

additional request types would enable the client and server to execute more specific functions related to creating, deleting, setting, and getting *shariables* and *shariable* information. Specifically, it would be useful for the server to handle the following message requests.

- ***/create/<vartype>/<varname>***. Creates a *shariable* named <varname> on the server of data type <vartype>.
- ***/exists/<varname>***. Returns a Boolean value to the client if the *shariable* currently exists on the server.
- ***/delete/<varname>***. Requests to delete a *shariable* from the server.
- ***/set/<varname>/<value>***. Updates a particular *shariable* to a new value.
- ***/get/<varname>***. Returns the latest value of the *shariable* from the server.
- ***/getall***. Returns a list of all the *shariables* on the server in response to the user pressing the "Import a Variable" button from the local NetScratch application.

7 User Study

After implementing NetScratch, I invited a group of six middle school children to spend an afternoon developing networked projects and providing feedback. While further testing should be done to fully evaluate NetScratch, observing how these children interacted with the system offered great insight for improvement. This chapter presents the demographics of the users, the goals for and structure of the workshop, and the observations made during the user study.

7.1 Demographics

The testers, four boys and two girls, attend a local middle school. As part of an after-school program at that middle school, they had been using Scratch for four months prior to the NetScratch user study. Consequently, they were already familiar with the Scratch environment, allowing me to focus on how they understood and interacted with the new NetScratch features.

7.2 Goals

The user study helped me assess whether the NetScratch networked features were engaging, accessible, evocative, and educational. Particularly, these were the primary questions I hoped to explore during the user study:

- ***What do they create?*** Like Resnick and Silverman, I felt that the diversity of projects created by the group would be a good indicator of success (12). In other words, NetScratch should enable children to create projects that grow out of their own personal interests.
- ***What do they learn?*** As mentioned previously, this technology is meant to "not only engage kids in constructing things, but also encourage (and support) them to explore the ideas underlying their constructions (12)."
- ***How do they collaborate?*** Because these new features are meant to enable children to work together on projects, it was important to observe the types of collaboration that were supported and inspired while using NetScratch.

- ***What do they recommend?*** NetScratch is a technology that should be engaging, exciting, and understandable for the users. Therefore, during the session, careful attention was paid to the users' specific feedback.

7.3 The Programming Session

First, I introduced *websensors* to the group, explaining that they could integrate information that people have put on the networks into their projects. They then spent 30 minutes experimenting and developing projects with *websensors*. Then I introduced *shariables*, emphasizing that now they could be the ones to *put* information on networks for other people to use. They spent the following 90 minutes working with both *websensors* and *shariables*.

While the structure of the workshop gave me a chance to invite specific feedback from the children, it mostly enabled me to observe how they interacted with the system. While observing the children programming and interacting with one another, I could infer what they found to be inspiring and/or confusing.

7.4 Observations

I learned a great deal from the user study, getting many ideas for improvement and a better understanding of how children interact with these networked features. This section is organized based on the primary goals I had for the user study.

7.4.1 What Did They Create Using *Websensors*?

Many of the *websensor* projects had a variety of sprites talking to each other, as shown in Figure 42. For example, sprites were programmed to say the word or joke of the day. This experimentation enabled children to explore the outputs of the *websensors*.

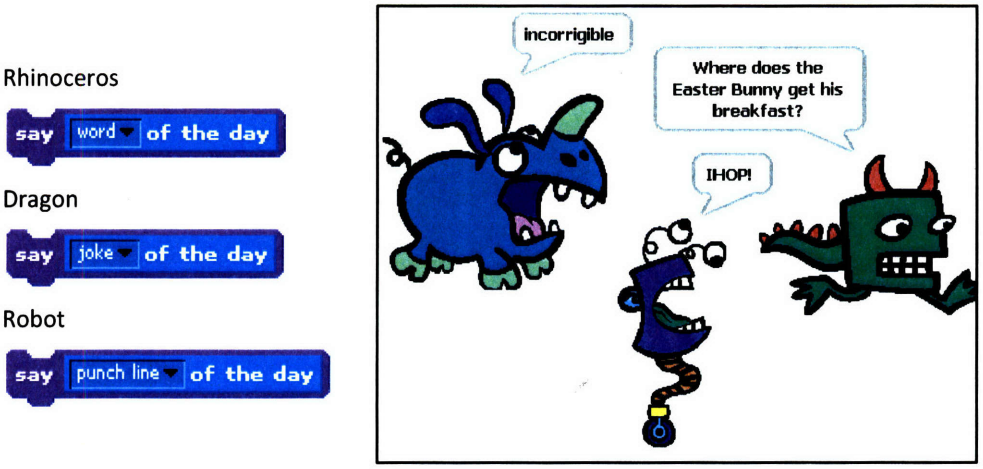


Figure 42: Project created by 13 year old boy, in which sprites say the values of the *websensors*

Some projects explored the effects of nesting *websensors* as shown in Figure 43. Other project ideas involved creating games with properties, such as the speed of a ball, that were affected by the current temperature and wind speed in a particular location.

Figure 43: Nested *websensors*

Based on project diversity, *websensors* did not seem to generate many project ideas that excited the children. While they did explore the real-time *websensor* data, they did not integrate the data into projects that leveraged the other capabilities of Scratch and their own interests. Instead, they were just experimenting with the new data available to them.

The lack of project diversity could be the result of a number of factors, including the limited time the children had to think about and create their projects, the lack of fully-integrated string functionality, or the lack of control of what types of web information they could integrate into their projects. For example, one girl wanted to include a daily horoscope in her project. This suggestion indicated that she might be more interested in *websensors* if she could control which *websensors* were available to her. A discussion on how to address these setbacks is presented in Section 8.1.

7.4.2 What Did They Create Using *Shariables*?

In contrast to *websensors*, *shariables* generated a lot of enthusiasm and project ideas. Some of the children developed a chat network in which they each created a string *shariable* used for message sending. Figure 44 presents a screenshot of the communications during the user study. They all continuously updated the *shariables* (named *attack*, *dragon*, *cole*, and *For all peoples*) at their own machines so that the rest of the group could see what they wrote. For example, in the image below, when the user executed **set cole to <a monkey!!!>**, the value was changed on every person's NetScratch application; and when someone on another computer executed **set attack to <really where?>**, the value was also updated on each NetScratch application, and the result is shown on the stage in the image below. These experiments gave the children ideas to create their own personalized chat systems, which is described more in Section 7.4.3.

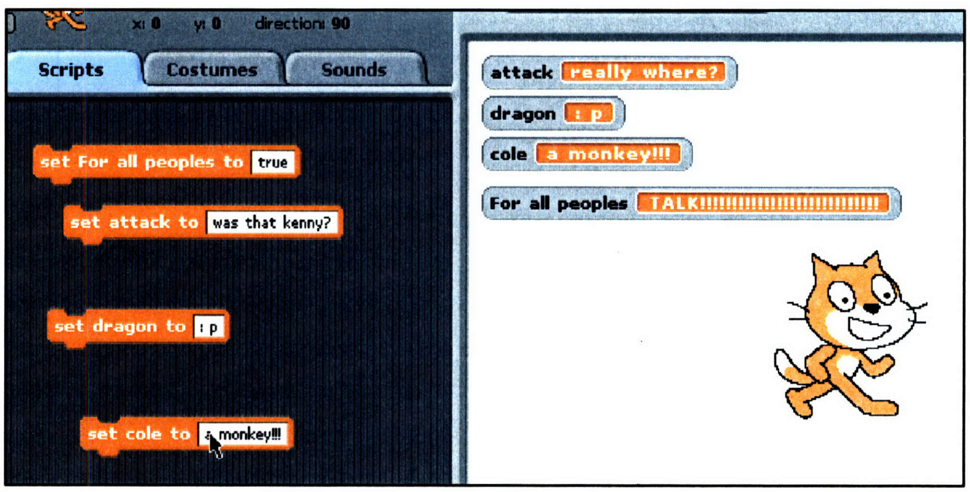


Figure 44: NetScratch used to generate a chat system

Another user *shariable* project explored having the loudness detected by one computer control a project running on another computer. On Computer 1, she created a *shariable* named *loud dragon* which was set to the loudness detected by the computer's microphone, which is reported through the **loudness** block. Another project, on Computer 2, contained a snowman whose position was determined by the value of the *loud dragon shariable*. Whenever she clapped her hands or made noise by Computer 1, the value of *loud dragon* would be set to the detected sound level, and Computer 2 would react by resetting the snowman's position (see Figure 45).

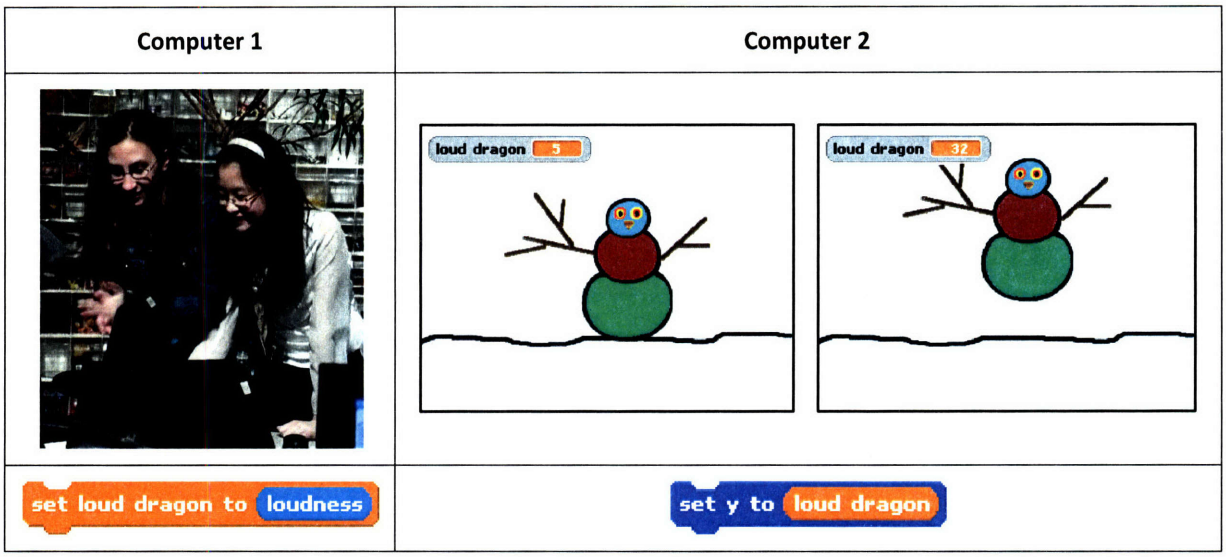


Figure 45: Sending loudness from one computer to another

Another user created a project in which a sprite would respond to direction commands that were sent from another machine. He created a *shariable* named *direction*, which could be set to up, down, left, or right from Computer 1, and Computer 2 contained a ball sprite that moved in the corresponding direction (see Figure 46). If he had more time, he wanted to extend his project to be a multiplayer networked Pong game.

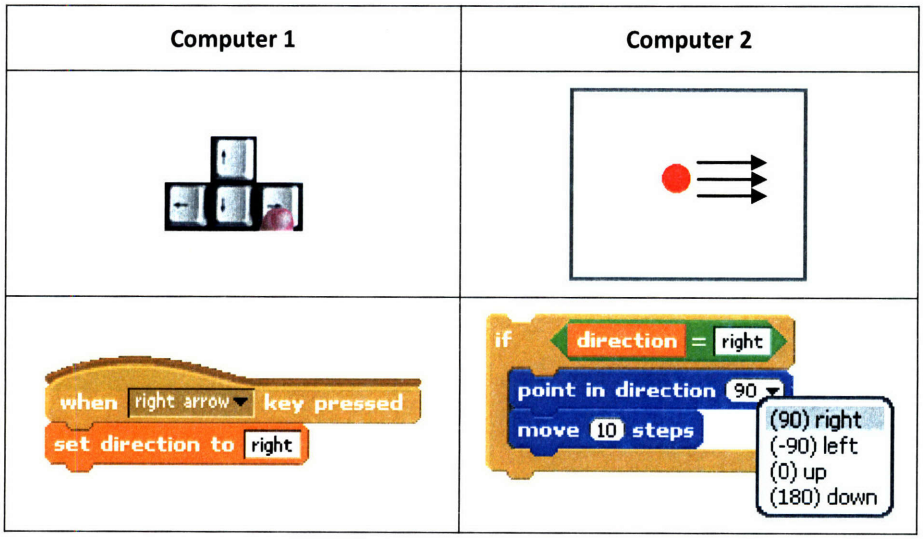


Figure 46: A *shariable* named *direction* is changed on one computer and controls a ball on another

Many of the children had ideas for projects that they would create if they had more time, such as a multiplayer role-playing environment. In general, the children were excited about *shariables* and felt empowered to use them in new and creative ways. I since learned from the woman who coordinates the after-school Scratch club at their school that these children continue to ask her when the NetScratch features will be made available in the official version of Scratch.

7.4.3 What Did They Learn?

Because the web is such a familiar technology to these children, they were able to easily grasp the underlying concepts needed to develop networked projects. Furthermore, the features in NetScratch enabled them to think about some of the issues that arise in developing a networked creation. Particularly, they demonstrated an understanding of the following concepts.

- ***Servers***. The children understood that there was a central computer (a server) that was directing *shariable* information from one computer to another. This understanding was demonstrated by their recommendations that that central "directing" computer be faster.
- ***Mashups***. They also understood that they could bring different types of data into their projects from the web, and combine them in their own ways, creating web mashups. Some of the children even suggested being able to sense pictures from the web, so they can use dynamically-changing pictures of cars that returned from a Google image search for "BMW" or of their friends from their MySpace profiles.
- ***Real-time data***. They understood that the real-time data their projects were relying on would not remain constant through the life of a project. To take advantage of this liveness, they created programs that would behave differently based on the incoming changing data, detected from either *websensors* or *shariables*.
- ***Persistence***. They understood the ideas of persistent data, as evidenced by their expectations that the *shariables* should remember the values that they were last set to, even after they closed and re-opened a NetScratch project.
- ***Scale***. To my surprise, the children also had an understanding of the potential scaling problems that would arise if the rest of the world started using these features. Particularly, they realized that if everyone were using *shariables*, there would eventually be too many *shariables* to sort

through. Thus, they addressed the potential scaling problems by coming up with suggestions presented in Section 7.4.5.

Beyond understanding the computer science, network, and design concepts, the children also experienced other types of understanding and empowerment. *Shariables* enabled the children in the workshop with a sense of being in control. One of the girls remarked, "*shariables* are like AIM [America Online Instant Messenger], except better because I'm in control." One of the boys offered to program a multiplayer networked game, but announced that because he would program the different characters, he would be the control center. This excitement about being in control suggested that the children felt empowered by the ability to control what is happening on one computer from another computer, and to have the projects that they create affecting other people.

7.4.4 How Did They Collaborate?

The introduction of *shariables* drove a new kind of collaboration in which the children were actively inviting others to be their collaborators. One boy even suggested, "Does this mean I can share a variable with someone in Norway?" Furthermore, throughout the session, some of the children would exclaim to the group, "who has a *shariable* I could use?" These interactions indicate that they were thinking of these new features as a new way to collaborate and socialize with people; they were creating their own social interaction tools, and thinking about how their projects could affect the projects of the people around them.

7.4.5 What Did They Recommend?

In general, the children found *shariables* exciting and inspiring, but found that the *websensors* could benefit from some improvements. The children provided the following feedback for how to improve the NetScratch features.

First, the children had a few suggestions for how to address the scaling problems that would arise as a result of many NetScratch users creating many *shariables*. Particularly, they offered the following suggestions for how to better sort and select *shariables*.

- ***Sharing within a particular group***. The children suggested that users could have a buddy list and their *shariables* would only be available to the people on that list. Users could also choose to share certain *shariables* with other groups, such as the members of a gallery on the Scratch website.
- ***Sharing with random people***. They also suggested that it would be good to add some randomness to the above suggestion, enabling users to use *shariables* from random people also. This feature would enable them to share with "someone in Norway," which they had been excited about when they were first introduced to *shariables* and realized that they could collaborate with people all over the world.
- ***Sending variables to people***. Another suggestion was that users should be able to "send" a *shariable* to particular people or projects, instead of relying on the receiver to actively import it.
- ***Sharing on the website***. They suggested that the sharing should happen on the Scratch website, which already supports friends and easy browsing of current Scratch users.
- ***Sharing with everyone***. Some wanted all *shariables* to be available to everyone, sorted based on the users that created them or users that use them. While this would create more clutter, it would also enable users to find anyone's interesting *shariables* and use them in their own projects.

As anticipated, the group wanted metadata associated with *shariables*. They wanted to know who created the *shariable* and who last updated it. This additional information became especially desired when the children "hijacked" (a term they used to describe someone mysteriously changing the value of a *shariable*) each other's *shariables*. They also wanted *shariables* to have read/write protections associated with them, assigning different users with different capabilities.

The group also had suggestions for the types of *websensors* they would include in their projects: horoscope, high score on online games, avatar pictures from MySpace, and Wikipedia information. This diversity suggested that *websensors* should be user-generated. A more detailed discussion of this extension is presented in Section 8.1.1.

8 CONCLUSIONS AND FUTURE WORK

Ultimately, the networked programming environment NetScratch enables children to create personally meaningful networked projects in a collaborative networked setting. When using NetScratch during the user study, children were inspired to create projects that they cared about, such as networked games and personalized chat systems. In designing these creations, children explored concepts about networks, distributed information, web mashups, real-time data, persistence, and scale.

While *websensors* and *shariables* did enable children to design networked interactive media in a collaborative way, there are some drawbacks to these features. Specifically, the current *websensors* are pre-determined and therefore did not necessarily reflect a user's personal interests. Additionally, the current *shariable* model lacks metadata and permissions. Furthermore, while adding strings provides richer data types in NetScratch, they are not yet fully integrated into the current Scratch framework. The remainder of this chapter addresses these drawbacks and discusses efforts that would make distributed and collaborative programming through NetScratch even more accessible to novice programmers.

8.1 NETSCRATCH IMPROVEMENTS

8.1.1 USER-GENERATED *WEBSENSORS*

While the few *websensors* that were available in NetScratch enabled children to think about dynamic networked data, the types of data available in the set of *websensors* did not engage children in projects and ideas that were specific to their interests. To address these limitations, the children suggested other types of real-time data that they would like to use in their projects; suggestions included horoscope reports, online game hits, high scores, and Wikipedia information. Undoubtedly, international NetScratch programmers would have completely different *websensor* suggestions from websites in many different languages. Ultimately, these observations led to the realization that children should feel empowered to seamlessly import any web data into their projects.

To address this need, the Scratch website could provide an assortment of *websensors* beyond the pre-defined collection currently available in NetScratch. Advanced users could add to the *websensor* assortment by using the 4-parameter pattern-matching algorithm presented in Section 6.1.2.

Furthermore, in addition to the current user interface, NetScratch would benefit from having an additional simple and seamless user interface, potentially a web browser add-on, particularly for creating user-generated *websensors*. In this way, any user can visit any website, and easily select information from the webpage to be fed into a NetScratch project.

Increasing the selection of *websensors* would enable NetScratch users to explore ideas and develop projects that are specifically meaningful to them.

8.1.2 *SHARIABLE* MANAGEMENT

The current NetScratch *shariable* infrastructure is simple in that anyone can create, import, view, or edit any *shariable*. While there are benefits to having a simple *shariable* experience, a more extensive *shariable* management system would be necessary if NetScratch became more widely used. In particular, there are a few user experience issues to be addressed:

- ***Permissions***. *Shariables* could have permissions associated with them that enable certain users with read, write, read/write, or no permissions. User suggestions for how these permissions should be distributed, such as specifying permissions for particular groups of people, are presented in Section 7.4.5.
- ***Metadata***. *Shariables* could also have metadata associated with them to enable a user to, for example, easily find all the projects using the particular *shariable* and the username of the person who last changed the value of the *shariable*. This feature would allow users to track how their *shariables* are used and disseminated.
- ***Search***. *Shariables* could become a part of the Scratch website experience by linking *shariables* to users. The *shariables* could have comments that explain how they are used. For example, I could explain that my *tammy mood shariable* gets updated every morning when I wake up, and the possible values are happy, sad, and thoughtful. Furthermore, there could be particular

shariables that are search-able on the Scratch website. For example, I could search for all friends whose current mood is sad, and create a project to cheer those people up.

- ***Clutter.*** Deleting *shariables* after extended periods in which they are not used would reduce the amount of clutter that users would need to sort through when attempting to find useful *shariables*. If a project using a deleted *shariable* is executed again after extended non-use, the *shariable* could automatically be recreated on the server.

8.1.3 Better Data Type Integration

Introducing strings into NetScratch suggests Scratch extensions for both handling new data types and fully integrating the new data types into the environment.

Particularly, introducing strings peripherally introduced a collection of other data types. For example, a Scratch user (in the **Scratch project of <Scratch user>** *websensor* block) is on one hand just a string representing the username, but is also a record data type that has Scratch friends, Scratch projects, and a location associated with it. Additionally, a zip code is a data type with a variety of information associated with it, including city and state names, weather conditions, wind speed, and time. To support seamless integration of these data types, NetScratch should support a set of standards for how particular data types appear in the environment. For example, the output of the **location of <Scratch username>** *websensor* block should return a value that could be used as input for the **temperature in <location>** *websensor* block. This is currently not the case, as the output of the **location of <Scratch username>** block reports a string representing a country name, while the input for the **temperature in <location>** block requires a number representing a zip code.

In addition to better handling of these new data types, strings in general could be better integrated into NetScratch, enabling users to more creatively use the *websensor* and *shariable* strings. Specifically, NetScratch would benefit from string manipulation capabilities, such as concatenation, parsing, and sorting. That way, users could create their own combinations of letters, words, sentences, and other data structures to integrate into their projects. For example, a hangman game might require the ability to break down a word into letters. Additionally, a user might want to parse a string when trying to dynamically use the hours part of a time represented as "6:12:26 pm."

Furthermore, strings and string variables could be more integrated into the NetScratch experience by enabling them to be used as broadcast messages or names of sprites. That way, one could programmatically broadcast and listen for dynamic messages, as well as dynamically interact with particular sprites based on the value of a string. For example, a sprite might rename itself to the result of the **Scratch friend of <Scratch username>** *websensor* block, and then other sprites could dynamically go to that sprite using the **go to <sprite>** block, with the **Scratch friend of <Scratch username>** block as an input parameter. This addition would make the "Web of Friends" sample project presented in Section 5.1.4 more dynamic since sprites can automatically reset their names to the result of dynamically-changing Scratch usernames. A further extension of string integration would enable users to input strings dynamically once a project is running. That way, a project could, for example, prompt a user for his or her Scratch website username, and then generate the list of that particular user's friends in real time.

Certainly, strings will become more integrated into the NetScratch experience as more *websensors* with string parameters are added. These additions would enable children to use strings in more dynamic and interesting ways. For example, similar to the YouTunes example presented in Section 2.5, a NetScratch project might involve searching for the most popular song listened to in 1999 (using, for example, a **most popular song in <year>** *websensor* block), and then use the result as input to a block that retrieves the lyrics (using a **get lyrics for <song name>** *websensor* block).

8.2 Further Evaluation

In order to assess how NetScratch could best enable users to create, collaborate, and learn, more extensive user studies should be done. Specifically, future user studies could focus on the following research questions:

- ***How do NetScratch users collaborate remotely?*** It would be useful to understand how children collaborate and communicate remotely while using *shariables*, as opposed to how they interact while they are in the same room.
- ***How do NetScratch interactions and creations change over time?*** Testing could focus on whether the NetScratch features enable children to learn over time and progressively explore

more advanced concepts. Additionally, one could observe whether the features encourage children to continually find new and creative ways to integrate networks into their creations.

- ***What do users learn when using NetScratch?*** Extended research on what children are thinking about when they use networked data would help guide future NetScratch extensions that encourage children to understand important computational and design concepts.
- ***Are websensors more engaging when user-generated?*** Evaluations could be done to see how children are engaged in integrating user-generated *websensors*. This research would help discover the best way to integrate data from the web into user creations.
- ***How should NetScratch be extended?*** Further user studies could focus on evaluating the most appropriate means through which to network Scratch projects. Following the advice of Resnick and Silverman, efforts should be made to "leverage what kids can do well, rather than focusing on what they can't," and, as a result, "developing programming languages and contexts that enable kids to do a lot with those basic elements. (12)" In that spirit, paying attention to how children interact with the current Scratch environment, specifically which features they already find engaging and intuitive, would help guide future NetScratch extensions.

8.3 BEYOND *WEBSENSORS* AND *SHARIABLES*

Beyond *websensors* and *shariables*, NetScratch can be extended to network other components of the Scratch programming environment. These changes would create a more seamlessly collaborative and dynamic Scratch programming environment in which children can connect quickly with people and websites that they care about.

Specifically, there are two ways in which the NetScratch environment could be expanded in the future to leverage networks. First, a wider variety of resources could be sensed from the web or shared among projects, such as the following:

- ***Image** shariables*. Because Scratch is a graphical multi-media environment, image development is often an important component of the design process. Thus, empowering children to share and update images on each other's projects would enable more collaboration in the development of connected projects.

- ***Image** websensors*. Similar to image *shariables*, *websensors* could expand the range of what is brought in from the web to include images, or perhaps even video, which could update in real-time as they do on the web. For example, an image sensor could return a random Google image search result for "snowman."
- ***Sound** websensors **and** shariables*. Like image *websensors* and *shariables*, these additions could detect sounds on the web or from another computer; then these dynamically-changing sounds could be played in real-time in a NetScratch project.
- ***Shared stage***. Users could each create their own sprites and add them to a shared stage, or a shared world. In this way, users can collaboratively contribute to a joint networked project in which their sprites interact.
- ***Shared scripts***. Script creation is at the heart of bringing Scratch creations to life. If scripts could be shared in real-time like *shariables*, then children could collaboratively develop scripts, and easily send them back and forth. This would enable children to develop dynamic pieces of code that could control other people's creations. For example, one user might create an *evolving dance routine* script that other users can easily see and use to make their own personalized sprites dance in new ways as the *evolving dance routine* updates.

Second, the environment could be expanded to better enable children to collaborate in the actual development process, as opposed to just sharing components of projects. This type of collaboration could be supported through an online authoring environment in which children could communicate about their ideas and creations. Some specific ways in which this type of collaboration can be supported in the future include:

- ***Collaboratively developing sprites***. A networked space for developing sprites would enable children to work collaboratively, each contributing to the sprite's costumes, sounds, and scripts. Once the sprite is created, children can use the sprite in their own separate projects. This functionality would empower children to work together on the program development and debugging process.
- ***Sharing entire Scratch programming space***. An entirely web-based collaborative development platform would enable children to work together on a shared screen. In this way, they could better understand how individual people approach the development of an entire Scratch project

and how different parts of the development process could be distributed among a group of programmers.

These additions would enable NetScratch users to immerse themselves in a fully-collaborative programming space.

8.4 Final Words

The work done in this thesis provides a foundation for making networked programming accessible to children. With NetScratch, children are able to think about and design networked creations that are personally meaningful, such as personalized chat systems and multiplayer online games. Using *websensors* to quickly and easily pull information from the web, children create their own web mashups and develop projects that depend on real-time data. With *shariables*, children can collaborate in creating networked projects that interact with each other through shared data. Designing these networked creations enables children to explore important computational and design ideas, while creating projects that they care about.

There are promising directions for future work that will extend the NetScratch environment to further enable the development of networked projects, as well as more effectively support a collaborative idea-development and project-creation experience. With the right tools and the right environment, children will be further empowered to design and implement networked creations, collaborate across networks, and think about powerful computational and design concepts.

REFERENCES

[1] *Distributed constructionism.* **Resnick, Mitchel.** Evanston, Illinois : International Society of the Learning Sciences, 1996. International Conference on Learning Sciences. pp. 280-284.

[2] Scratch: Imagine, Program, Share. *Scratch website.* [Online] [Cited: April 5, 2007.] http://scratch.mit.edu.

[3] *Lifelong Kindergarten website.* [Online] [Cited: April 5, 2007.] http://llk.media.mit.edu/.

[4] The Media Lab: Inventing a Better Future. *The Media Lab website.* [Online] [Cited: May 6, 2007.] http://media.mit.edu/.

[5] **Papert, Seymour.** Mindstorms: Children, Computers, and Powerful Ideas. New York : Basic Books, 1980.

[6] **Resnick, Mitchel, Kafai, Yasmin and Maeda, John.** A Networked, Media-Rich Programming Environment to Enhance Technological Fluency at After-School Centers in Economically Disadvantaged Communities. 2003. Proposal to the National Science Foundation (funded 2003-2007).

[7] *Squeak website.* [Online] [Cited: April 26, 2007.] http://www.squeak.org/.

[8] Welcome to LCSI. *Global Leader is Constructivist Educational Technology website.* [Online] [Cited: April 6, 2007.] http://www.microworlds.com/.

[9] **Kay, Alan.** Computers, Networks, and Education. *Scientific American.* 1991, Vol. 265, pp. 100-107.

[10] *Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers.* **Kelleher, Caitlin and Pausch, Randy.** Carnegie Mellon University : June 2005, ACM Computing Surveys, Vol. 37, pp. 88-137.

[11] **Salomon, Gavriel.** *Distributed Cognitions: psychological and educational considerations.* Cambridge, UK : Cambridge University Press, 1993.

[12] *Some Reflections on Designing Construction Kits for Kids.* **Resnick, Mitchel and Silverman, Brian.** New York, NY : ACM Press, 2005. Interaction Design And Children. pp. 117 - 122.

[13] *NetMorph - an intuitive mobile object system.* **Umezawa, Masashi, et al.** : 2003. First Conference on Creating, Connecting and Collaborating through Computing (C5). pp. 32-39.

[14] *The agentsheets behavior exchange: supporting social behavior processing.* **Repenning, Alexander and Ambach, James.** Atlanta, Georgia : ACM Press, 1997. Conference on Human Factors in Computing Systems. pp. 26-27.

[15] *Easy creation of network-based interactive animations using Islay.* **Okamoto, Shusuke, et al.** 2005. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing. pp. 412-415.

[16] **Bruckman, Amy.** *MOOSE Crossing: Construction, Community, and Learning in a Networked Virtual World for Kids.* Program in Media Arts and Studies, MIT : Massachusetts Institute of Technology, 1997. Doctoral Thesis.

[17] Pipes Overview. *Yahoo Pipes website.* [Online] [Cited: April 26, 2007.] http://pipes.yahoo.com/pipes/docs?doc=overview.

[18] **Feinberg, David.** *Broadcast-Based Communication in a Programming Environment for Novices.* Electrical Engineering and Computer Science : Massachusetts Institute of Technology, 2004. Masters Thesis.

[19] *The Hook-ups initiative: how youth can learn by creating their own computer interfaces and programs.* **Millner, Amon.** New York, NY : ACM Press, December 2003, ACM SIGGROUP Bulletin, Vol. 24, pp. 85-89.

[20] *Google Gadgets website.* [Online] [Cited: April 19, 2007.] http://www.google.com/webmasters/gadgets/.

[21] Tamagotchi Connection. *Tamagotchi website.* [Online] [Cited: April 29, 2007.] http://www.tamagotchi.com/.

[22] Welcome to Webkinz. *Webkinz website.* [Online] Ganz. [Cited: April 29, 2007.] http://www.webkinz.com/.