# MOOSE Crossing: Construction, Community, and Learning in a Networked Virtual World for Kids

by

Amy Susan Bruckman

Bachelor of Arts, Physics (1987)
Harvard University

Master of Science in Visual Studies, Interactive Cinema (1991)
Massachusetts Institute of Technology

SUBMITTED TO THE PROGRAM IN MEDIA ARTS AND SCIENCES,
SCHOOL OF ARCHITECTURE AND PLANNING
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY AT THE MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1997

*Author* ______________________________________________
Program in Media Arts and Sciences
March 17th, 1997

*Certified by* ______________________________________________
Mitchel Resnick
Associate Professor of Media Arts and Sciences
Massachusetts Institute of Technology

*Accepted by* ______________________________________________
Stephen A. Benton
Chair, Departmental Committee on Graduate Students
Program in Media Arts and Sciences
Massachusetts Institute of Technology

# MOOSE Crossing: Construction, Community, and Learning in a Networked Virtual World for Kids

by

Amy Susan Bruckman

SUBMITTED TO THE PROGRAM IN MEDIA ARTS AND SCIENCES,
SCHOOL OF ARCHITECTURE AND PLANNING ON MARCH 17, 1997,
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

**ABSTRACT**

In research about the Internet, too much attention is paid to its ability to provide access to information. This thesis argues that the Internet can be used not just as a conduit for information, but as a context for learning through community-supported collaborative construction. A "constructionist" approach to use of the Internet makes particularly good use of its educational potential. The Internet provides opportunities to move beyond the creation of constructionist tools and activities to the creation of "constructionist cultures."

These issues are explored through a specific example: MOOSE Crossing, a text-based virtual world (or "MUD") designed to be a constructionist learning environment for children ages 8 to 13. On MOOSE Crossing, children have constructed a virtual world together, making new places, objects, and creatures. Kids have made baby penguins that respond differently to five kinds of food, fortune tellers who predict the future, and the place at the end of the rainbow—answer a riddle, and you get the pot of gold.

This thesis discusses the design principles underlying a new programming language (MOOSE) and client interface (MacMOOSE) designed to make it easier for children to learn to program on MOOSE Crossing. It presents a detailed analysis, using an ethnographic methodology, of children's activities and learning experiences on MOOSE Crossing, with special focus on seven children who participated in a weekly after-school program from October 1995 through February 1997.

In its analysis of children's activities, this thesis explores the relationship between construction and community. It describes how the MOOSE Crossing children motivated and supported one another's learning experiences: community provided support for learning through design and construction. Conversely, construction activities helped to create a particularly special, intellectually engaging sort of community. Finally, it argues that the design of all virtual communities, not just those with an explicitly educational focus, can be enhanced by a constructionist approach.

Thesis Supervisor: Mitchel Resnick
Associate Professor of Media Arts and Sciences,
Massachusetts Institute of Technology

This work was performed at the MIT Media Laboratory. Support for this work was provided by AT&T, British Telecom, Digital Equipment Corporation, Gruppo Grauso, Interval Research, The LEGO Group, The National Science Foundation (grant 9358519-RED), the Media Lab's News in the Future Consortium, Nintendo, and Telecom Italia.

## Doctoral Dissertation Committee

*Thesis Advisor* ______________________________________________
Mitchel Resnick
Associate Professor of Media Arts and Sciences
Massachusetts Institute of Technology

*Thesis Reader* ______________________________________________
Pavel Curtis
Principal Architect
PlaceWare, Inc.

*Thesis Reader* ______________________________________________
Henry Jenkins
Director, Program in Film and Media Studies
Massachusetts Institute of Technology

*In memory of*
*my grandmothers:*

Florence Fox Bruckman
Norma Brodney Cohen

## Acknowledgments

A number of people deserve not just thanks, but direct credit for some of the work this thesis represents. In particular, much of the intellectual work of this thesis took place in conversations with my advisor, Mitchel Resnick. Mitch has been not just a great advisor, but a great friend.

A number of MIT Undergraduates worked on the project under the auspices of MIT's Undergraduate Research Opportunities Project (UROP). The UROP program is one of the best parts of being at MIT. Working on the MacMOOSE client in chronological order were: Greg Hudson, Adam Skwersky, Steve Tamm, Jon Heiner, and Drew Samnick. Steve Shapiro handled the port to JavaMOOSE. Special thanks are due to Steve Tamm for sticking with the project over time, and working hard to support his successors on the development team.

On the server and people side, Albert Lin helped with the launch of MediaMOO, and with early work on MOOSE Crossing. Trevor Stricker helped with the MOOSE language design, wrote the server modifications for data collection, and kept the kids laughing. Austina Vainius helped with every aspect of the project, including writing documentation, designing objects for the MOOSE Crossing world, helping with revisions of the MOOSE language and client based on feedback from kids, and taking terrific notes on the kids' learning experiences. She has spent countless hours working with children online. One visitor to a MOOSE-Crossing classroom commented that telling the kids he had met Austina was like admitting he knew Elvis.

A number of people gave thoughtful comments on drafts of this thesis. I'd like to single out Danny Bobrow and Carol Strohecker in particular, for giving me some new ways to think about things. Other extremely helpful comments came from Andy Begel, Aaron Brandes, Justine Cassell, Mark Guzdial, Greg Kimberly, Jon Orwant, David Shaffer, Vanessa Stevens, Michael Travers, Austina Vainius, and Terry Winograd. My advisor Mitch Resnick read each chapter of this document two to four times, carefully each time. My thesis readers Pavel Curtis and Henry Jenkins provided lots of helpful feedback.

Pavel Curtis wrote the MOO software, on top of which MOOSE is built. He made a number of modifications to MOO to make MOOSE possible. Special thanks to Pavel for being a diligent thesis reader despite having other tiny little things like a startup company to worry about.

Henry Jenkins is an inspiration in all of his work. I thank Henry for reminding me to believe not just in active readers, but in the basic creativity and intelligence of people.

Sherry Turkle taught me how to think about how people relate to media. It was Sherry who encouraged me that thinking about MUDs and people's relationships to them and within them was a topic worthy of intellectual study.

Brian Silverman gets a gold star for being there to help me pull the technical pieces together. One afternoon in 1994, I tickled Boo Boo Bear and then went to get a cup of coffee while I waited for him to laugh. That was when I realized we had some serious performance problems. Brian helped me tear the system apart and put the pieces back together more efficiently.

Glorianna Davenport, my master's thesis advisor, gave me the chance to be at the Media Lab in the first place, and has continued to support my work.

I'd like to thank members of the Media Lab's narrative-intelligence reading group for helping to create a greater sense of intellectual community at the lab. Other communities that have helped sustain my MIT experience are the AI Lab dance group, and the thirsty-magic gang.

Michael Maier made the nifty moose icon used on the MacMOOSE application and other parts of the MOOSE Crossing project.

Jon Callas, Randy Farmer, and Dean Tribble contributed ideas to the design of the MOOSE language. Developing a friendship with Randy online has probably been my best personal experience with use of this medium. I've benefited greatly from all of Randy's experience from Habitat and beyond.

I'd like to thank Brenda Laurel, Howard Rheingold, and Sandy Stone for existing. And for having more faith in my work than I deserve.

Thanks to the members of MediaMOO and MOOSE Crossing for making these projects happen. On MediaMOO, special thanks to members of MediaMOO's Membership Advisory Committee for all their time, especially Michael Day, Randy Farmer, Beth Kolko, and Diane Maluso. Thanks to Tari Fanderclai and Greg Siering for organizing The Tuesday Café. Finally, thanks to all of MediaMOO's volunteer janitorial staff, especially Judy Anderson, Gustavo Glusman, and Seth Rich.

On MOOSE Crossing, special thanks are due to the children who participated in the Media Lab's MOOSE Crossing after-school program, and to their parents—for letting me study their children, and for transporting them to and from the lab each week for almost a year and a half. I'd also like to thank

residents of The Pond area of the lab for putting up with the noise and the litter of empty hot-chocolate cups each Monday.  Special thanks are also due to the teachers who brought MOOSE Crossing into their classrooms, especially BJ Conn, Andrea Martin, and Richard Smyth. I hope it was worth the effort.

Thanks to members of the Epistemology and Learning Group for being such a great group of people to learn and play with. Special thanks to my officemate David Cavallo for many long conversations about the meaning of it all, and for putting up with my mess.

Thanks to Jon Orwant for moral and emotional support, emergency Perl scripting, and sharing his Nintendo 64.

Finally, thanks to my family: Mom and Bernie; Dad and Mari, Danielle, Alicia, Rachael, Lucy, and Yikes. Not to mention Gilda and Judy; Betty, Midge, and Drew. For everything.

## Table of Contents

Abstract....3

Acknowledgments....7

Table of Contents....10

1. Introduction: MOOSE Crossing....13
   - 1.1 An Evening at The Crossing....13
   - 1.2 Construction and Community....16
   - 1.3 What is a MUD?....19
   - 1.3.1 MUD History....19
   - 1.3.2 Personas....20
   - 1.3.3 Places....22
   - 1.3.4 Objects....25
   - 1.4 Prior Work: MediaMOO....27
   - 1.5 The MOOSE Crossing Project....27
   - 1.6 Outline....30

2. The Day After Net Day: Approaches to Educational Use of the Internet....33
   - 2.1 The Hype and the Reality....33
   - 2.2 Distance Education....35
   - 2.3 Information Retrieval....37
   - 2.3.1 Children Accessing Controversial Information....38
   - 2.4 Knowledge-Building Communities....40
   - 2.4.1 Global Science....40
   - 2.4.2 CSILE....42
   - 2.4.3 Professional Communities....43
   - 2.4.4 Real-Time Writing....46
   - 2.5 Technological Samba Schools....47

3. Designing Enabling Technologies: The MOOSE Language and the MacMOOSE Client ....53
3.1 The Need for a New Language ....53
3.2 The Design of the MOOSE Language....56
3.2.1 A Gently-Sloping Learning Curve ....58
3.2.2 Intuitive Simplicity Versus Formal Elegance ....59
3.2.3 Be Forgiving....64
3.2.4 Leveraging Natural-Language Knowledge ....65
3.2.5 Avoid Non-Alphanumeric Characters....68
3.2.6 Make Essential Info Visible and Easily Changeable ....72
3.2.7 It's OK to have Limited Functionality....74
3.2.8 Hide Nasty Things Under the Bed ....75
3.2.9 A Design Philosophy....77
3.3 The Need for a New Programming Environment ....78
3.4 The Design of the MacMOOSE Client ....80
3.4.1 A Tour of MacMOOSE....81
3.4.2 Equal Access for Few Versus Unequal Access for Many ....88
3.5 Designing Empowering Technologies ....89

4. Collaborative Learning Strategies: Storm's Weekend with Rachael....91
4.1 Storm's Weekend with Rachael....91
4.2 Friday: Diving In ....93
4.3 Saturday: Independent Progress ....102
4.4 Sunday: More Mutual Reinforcement....105
4.5 Monday: Camaraderie Combats Frustration....108
4.6 Tuesday: Collaboration ....110
4.7 Seven Months Later: Meeting Face to Face....114
4.8 Conclusion: Integrating Technological and Social Contexts....116

5. Community Support for Construction ....119
5.1 Uzi's Frustration ....119
5.2 Pilot Study: Programming for Fun ....120

5.3 "Did You Really Make This?": The Importance of Role Models ..... 124
5.4 Situated, Ubiquitous Project Models ..... 126
5.5 Emotional and Technical Support ..... 128
5.6 An Appreciative Audience ..... 135
5.7 Local Community and Online Community ..... 145
5.8 An Extended Example: Lady's First Script ..... 149

6. Constructionist Culture ..... 159
6.1 A Felicitous Type of Community ..... 159
6.2 "Television Fans and Participatory Culture" ..... 159
6.3 Objects of Construction ..... 160
6.4 Worlds Made by Their Inhabitants ..... 166
6.5 Sharing Scarce Resources ..... 168
6.6 Believing in Users ..... 170
6.7 Construction and Community ..... 172

7. Conclusion: Constructionism and Virtual Communities ..... 175
7.1 Open Research Questions ..... 175
7.1.1 The Social Implications of Distributed Systems ..... 175
7.1.2 The Cognitive Implications of Graphical Media ..... 176
7.1.3 Gender, Technology, and Learning Styles ..... 178
7.2 Contributions ..... 183

Appendix: Five Children's Creations ..... 187
Goofy ..... 188
Mouse ..... 189
Angela ..... 212
Werdna ..... 213
Rowena ..... 219

Bibliography ..... 224

# 1. Introduction: MOOSE Crossing

## 1.1 An Evening at The Crossing

It’s early evening on a Thursday in September, 1996. Hermes[1] (boy, age 9) is working on his Magic Subway Station. Miranda (girl, age 11)[2] is in her room.

Miranda sees:

```
You sense that Hermes is looking for you in Magic Subway Station.
Hermes pages, 'Hi! can i come over?'
```

Hermes sees:

```
All of a sudden, you hear a clap of lightning and a rumble of
   thunder.  You look into the sky, and see Miranda's message written
   in the stars!
Miranda pages, 'Ok '
```

Hermes types: `join Miranda`

He sees:

```
Miranda's room
A bright room in what look[3] like the center of the sun, all around
you there is a bright yellow light.  A green hand emerges from the
wall and shoves a pair of sunglasses on your face.  The sun glasses
turn the bright light into a dull glow.  You can barely make out a
arm chair and sofa.  The floor is covered in a gold rug.
   Obvious exits: ..crazy........Miranda's Crazy Gadgets shop
                  ..fly..........Travel Agency
                  ..fortune......The Fortune Tellers Caravan
                  ..in...........< Paradise Island >
                  ..stroll.......The GreenHouse
   You see Juju, Huey, and Duey here.
   Miranda is here.
```

They both see:

```
Hermes shatters the air with his explosive arrival.
Google arrives, following Hermes.
Miranda says, 'Hi!'
Miranda says, 'How old are you?'
Hermes says, '9'
Hermes says, 'you?'
Miranda says, 'oh, I'm 11, I'm Mouse's sister'
```

[1] I have changed the children’s character names to protect their identities. Although their character names are already aliases, changing them gives an added level of privacy. I tried to pick names which convey the same flavor as the original, but are not obviously identifiable as the same person. People who work in and around the area of The Media Lab known as “The Pond” contributed many creative name suggestions.

[2] Miranda started participating in MOOSE Crossing when she was ten years old. At the time of this writing, she is still an active member and is now twelve. Many children have participated for more than a year. I will indicate their age with a range when I am refering to their participation in general terms. When I quote a specific incident or completed project, I will list their age at the time the event took place or the project was created. As a result, some children’s ages will be listed differently in different sections of this thesis.

[3] Miranda presumably meant to use the word “looks” here. The children’s conversations and descriptions of their creations are presented unedited and uncorrected (except where otherwise noted, and where editing was necessary to protect their privacy.)

Miranda types: `become Athena`

```
Miranda waves her hand over her entire body.  Every place her hand
   passes, changes into a new person!
Hermes says, 'mind having the magic subway connected to paradise
   island?'
```

Miranda has built a vacation resort called Paradise Island where you can swim and build a summer home. She also made a travel agency and car rental agency to arrange trips there.

Hermes types: `become Cloudstreamer`

```
Hermes suddenly floats up in the air and is engulfed in light.  The
   old form melts away and is magically transfomed into an eagle.
   It's Cloudstreamer!
Athena says, 'Fine with me, but would you mind connecting it to my
   Travel Agency? you don't have to’
Cloudstreamer says, 'ok, but you have to help connect it.'
```

Miranda/Athena and Hermes/Cloudstreamer work to connect his magic subway station to her travel agency. While they are working, he notices that a new member, Scribbles (girl, age 13), has connected.

Scribbles sees:

```
You sense that Cloudstreamer is looking for you in Magic Subway
   Station.
Cloudstreamer pages, 'hi! i'm new too'
```

Cloudstreamer sees:

```
You sense that Scribbles is looking for you in MOOSE Crossing.
Scribbles pages, 'hi! can I join you? Love the name!'
```

Cloudstreamer has to go to dinner, but says he’ll be back in half an hour. Not long after:

```
Cloudstreamer pages Scribbles, 'im baaack!!come on over!'
```

Scribbles types: `join Cloudstreamer`

She sees:

```
Magic Subway Station
Welcome to The Magic Subway Station!  Please select your destination
and step through the magic portal!
   Obvious exits: ..CR..........Crossroads
                  ..EA..........Emerald Apartments
                  ..HC..........Home in the Clouds
                  ..NM..........North Main Street
                  ..PI..........< Paradise Island >
                  ..RF..........Redwood Forest
                  ..TA..........Travel Agency
   You see Google here.
   Cloudstreamer is here.

Scribbles teleports in.
Scribbles says, 'Hi there!'
Cloudstreamer says, 'hi'

Cloudstreamer types:   look Scribbles
He sees:
You see a brand new MOOSE Crossing member.
She is awake and looks alert.

Scribbles grins
Cloudstreamer says, 'wach this'
Scribbles says, 'this is my very first time on.  How about you?'
Cloudstreamer says, 'probably my 10th'

Cloudstreamer types:   become Widget

Cloudstreamer is suddenly covered by robot mechanics the size of mice
   carrying wrenches and drills and screwdrivers.  Body parts fly
   everywhere!  (Grownups, don't read this!)  Yikes!  Duck!  There
   goes an eyeball!  The metal mice vanish, and before you now stands
   ... drumroll, please! ... the one and only Widget!!!
Widget says, 'good. huh'
Scribbles jumps up and down.
Scribbles says, 'Yes.  AWESOME!'
Scribbles says, 'how'd you do it?'
Widget says, 'wanna do it?'
Scribbles says, 'it might be kinda fun.'
```

Hermes/Cloudstreamer/Widget shows Scribbles how to describe herself, and how to have multiple personas. He's just built a home for himself on Sparky II, a mobile community made by Rufus (boy, age 12). He shows her how to build a home there too. Soon he has to go to bed. After he leaves, Scribbles works more on her description:

```
A small, cinnamon-colored ball of short, fluffy fur and green
splendor stares up at you with twinkling eyes and a vivacious smile.
A mouse, you realize after a moment - a mouse wearing a tiny, white
silk shirt and close fitting green velvet breeches.  She bows deeply
and doffs a triangular, green hat with a bobbing golden plume pinned
to the side.  You can't help wondering where she obtained this odd
attire in such a minute size.  There is no doubt that it is of the
finest make.  Graceful vines, embroidered in thread-of-gold, climb up
the sides of her breeches and adorn the neck and cuffs of her shirt.
Tiny boots of a soft brown velvet lace up to her fuzzy knees.  Her
hat, which she has now returned to her head, is a perfect miniature
of those you have seen in full size.  Yet there is something else
about her something that draws your attention more than her odd dress
or her tiny size.  It is something in the way she carries herself,
the way she looks expectantly up at you with her paws on her hips and
her booted feet set slightly apart.  She looks like someone who has
been through a great deal, and always come out on top.  Like someone
who knows their destiny
```

In real life, Miranda is in Massachusetts, Hermes is in Florida, and Scribbles is in Maryland. They have never met face to face. They are participating in a text-based virtual world (or "MUD") for kids called MOOSE Crossing. They are imagining new places and objects, and creating them with words and programs. In their spare time for fun they are reading, doing creative writing, and writing computer programs. Through these activities, they are learning in a self-motivated, self-directed, peer-supported fashion.

This research examines how the Internet can be used not just as a conduit for information, but as a context for learning through community-supported collaborative construction. In *Mindstorms,* Seymour Papert has a vision of a "technological samba school." At samba schools in Brazil, a community of people of all ages gather together to prepare a presentation for carnival. "Members of the school range in age from children to grandparents and in ability from novice to professional. But they dance together and as they dance everyone is learning and teaching as well as dancing. Even the stars are there to learn their difficult parts" (Papert 1980). People go to samba schools not just to work on their presentations, but also to socialize and be with one another. Learning is spontaneous, self-motivated, and richly connected to popular culture. Papert imagines a kind of technological samba school where people of all ages gather together to work on creative projects using computers. MOOSE Crossing is an attempt to realize that vision.

## 1.2 Construction and Community

The central claim of this thesis is that community and construction activities are mutually reinforcing. Working within a community helps people to become better dancers/programmers/designers and better learners. Conversely, working on design and construction projects together helps to form a strong, supportive community.

This interdisciplinary thesis is addressed to several overlapping communities: computer scientists, educators, educational researchers, researchers in computer-supported cooperative work (CSCW), designers of virtual communities, and members of virtual communities. The underlying research draws ideas from each of these communities, and is also strongly influenced by concepts from anthropology, cultural studies, gender studies, psychology, and sociology. Different facets of this central theme about construction and community need to be highlighted for each of these communities.

From an education perspective, MOOSE Crossing shows how constructionism can be supported by community. "Constructionism" is a term first coined by Seymour Papert as an expansion of Jean Piaget's "constructivism." Papert writes:

> We understand "constructionism" as including, but going beyond, what Piaget would call "constructivism." The word with the v

> expresses the theory that knowledge is built by the learner, not supplied by the teacher. The word with the n expresses the further idea that this happens especially felicitously when the learner is engaged in the construction of something external or at least shareable... a sand castle, a machine, a computer program, a book. This leads us to a model of using a cycle of internalization of what is outside, then externalization of what is inside. (Papert 1991)

Mitchel Resnick expands on this notion, writing “They might be constructing sand castles, LEGO machines, or computer programs. What’s important is that they are actively engaged in creating something that is meaningful to themselves or to others around them” (Resnick 1994). This thesis gives new attention to those “others around them” and the essential role they play in the learning process.

While the term “constructionism” was only coined in the 1980s, research in this spirit to design computerized tools to help children learn by making things began in the 1960s. However, tools alone are not enough. All too often, tools designed to empower learners end up being used in the same old disempowering ways. The educational philosophy and spirit in which the tools were designed needs to be distributed together with the software or hardware. It’s not possible to communicate these ideas through a software interface, or through a manual no one will read. Tools can be effectively constructionist only when they are embedded in a constructionist culture. Constructionism works best when it is situated in a supportive community context. A kind of constructionist culture often emerges when the tool’s designer is present to help grow a community of users, but this spirit usually fails when dissemination is tried on a larger scale. Computer networks can be used to help create and spread constructionist cultures.

Papert imagined his technological samba school as a physical place. Virtual communities can be used to create such a “place” on the Internet. Both real and virtual places can help to create a supportive context for human activity and social interaction. You can also get the best of both worlds by using a virtual community to link many small face to face communities (such as after-school programs) into a larger whole.

At the time of this writing, an increasing number of educators are coming to realize that The Net is not an educational panacea. The process of getting computers hooked up to the Internet has been increasingly politicized—on Net Day in California in 1996, the President and Vice President of the United States themselves rolled up their sleeves and pulled red, white, and blue cable through a school. The day after Net Day, however, many educators found themselves wondering: Now we have net connections, but what exactly are we supposed to do with them? What is the real educational potential of this technology? I hope classroom educators will find this research to be one useful model. Chapter Two, "The Day After Net Day: Approaches to Educational Use of the Internet," reviews various approaches to using the net as an educational tool. Most applications focus on information—receiving it, retrieving it, or sharing it. MOOSE Crossing focuses on seeing the Net as a place for learning through community-supported collaborative construction.

From a computer science perspective, MOOSE Crossing demonstrates the viability of end-user programming, and the value of community support for learning to program. As technologies increasingly surround the every day lives of ordinary people, it remains an open question how much control people will have over those technologies. On MOOSE Crossing, children as young as seven years old have been able to write simple computer programs; thirteen-year-olds have been able to write moderately complex ones. I designed the MOOSE programming language and MacMOOSE client interface to help make this possible. Chapter Three, "Designing Enabling Technologies: The MOOSE Language and the MacMOOSE Client," reviews their design. MOOSE is a special-purpose language designed for only one application: helping kids to make creative, cute objects in a text-based virtual reality environment.

While good, expressive tools are essential, I believe the real key to expanding what users can accomplish is community support. Computer networks can enable that essential support. Researchers in computer-supported cooperative work (CSCW) have long recognized this potential. Readers from a CSCW background will be most interested in Chapter Four, "Collaborative Learning Strategies: Storm's Weekend with Rachael" and Chapter Five, "Community Support for Construction." The detailed data collected on children's learning experiences on MOOSE Crossing sheds new light on the ways in which community support can help motivate and support an individual's work.

From a virtual communities perspective, MOOSE Crossing and MediaMOO (an earlier community I designed) demonstrate the benefits of letting users be creators, not just inhabitants, of virtual worlds. Cyberspace is not Disneyland—it's not a place for people to obediently wait on line, and then sit strapped into a seat viewing what is presented. It's a place for people to participate and create (Bruckman 1995). It's essential to view users as creators, not merely recipients, of content. Constructionist ideas are of central importance to the design of virtual communities. Chapter Six, "Constructionist Culture," argues that constructing things together helps to create a particularly special, supportive sort of community.

At the intersection of these various communities are educators designing applications of virtual communities. To that community I would like to plead: please don't have virtual classes where students sit behind virtual desks and teachers write on virtual blackboards. To do so combines some of the worst aspects of both traditional pedagogy and virtual worlds. Children learn better by working on personally meaningful projects than by being lectured to. This is even more true in the virtual world than in face to face classrooms. A constructionist pedagogy makes better use of the affordances of MUDs. Like any medium, MUDs have affordances and limitations: things they are good for and things they are not good for. They should not be used for every application. They are lousy places to conduct traditional classroom-like education. They are superb places for constructionist learning.

## 1.3 What is a MUD?

### 1.3.1 MUD History

MOOSE Crossing is a "MUD." For historical reasons, "MUD" stands for "Multi-User Dungeon." The first MUDs were networked multi-player dungeons and dragons games. Since many current applications of MUD technology have moved far from their origins as violent games, some people prefer to say that the "D" stands for some other word such as "Domain" or "Dimension." I prefer to use the historically correct word.

MUDs have their origins in text-based adventure games. The first such game, "ADVENT," was written by Will Crowther and Don Woods around 1967 (Crowther 1992). In the late 1970s, Roy Trubshaw and Richard Bartle at Essex University in Great Britain put a multi-user text-based world up on the Arpanet and called it "MUD" (Bartle 1990). (It's commonly referred to now as "MUD1" to avoid confusion with the generic use of the term.) At around the same time, Alan Klietz made a similar program called "*E*M*P*I*R*E*," which was soon renamed "Scepter of Goth" (Klietz 1992). All MUDs for the next ten years would be modeled on Dungeons and Dragons games where players try to kill monsters and find magic treasure.

In 1989, a graduate student at Carnegie Mellon University named James Aspnes decided to see what would happen if he removed the monsters and the magic swords. In most MUDs up until this time, extending the virtual world was a privilege reserved for either the owners of the computer the game ran on, or those who had dedicated hundreds of hours of playing time and succeeded in winning the game by completing all of its quests. Aspnes decided to let everyone extend the virtual world. He removed many of the combat-oriented commands from the MUD software he was modifying, and the resulting code was much more compact. Consequently, he called his software "TinyMUD." The choice of names is ironic, because TinyMUD soon became huge. With few controls on who could build how many rooms, the database grew out of control. Students from several universities built detailed replicas of their campuses. The model of MIT included a detailed copy of The Media Lab. In the lab's cube auditorium was the set of a play then being performed there, and a negative review of the play was posted on the wall. The original TinyMUD soon got too big for the computer it was running on, and was shut down.

TinyMUD was a more egalitarian and pacifist community than its predecessors. I asked James Aspnes if those ideals came from himself and the community's founders. What had been his design goals? He replied:

> You raise an interesting question about the ideals of the TinyMUD community coming from the few founding members. Most adventure-style games and earlier MUDs had some sort of scoring system which translated into rank and often special privileges; I didn't want such a system not because of any strong egalitarian ideals (although I think that there are good egalitarian arguments against it) but because I wanted the game to be open-ended, and any scoring system would have the problem that eventually each player would hit the maximum rank or level of advancement and have to either abandon the game as finished or come up with new reasons to play it. This approach attracted people who liked everybody being equal and drove away people who didn't like a game where you didn't score points and beat out other players (I did put in a "score" command early on since almost everybody tried it, but most players soon realized that it was a joke). I think that this effect created a kind of natural selection which eventually led to the current egalitarian ideals. I like the egalitarianism, but it wasn't my original goal. (Aspnes 1992; Bruckman 1992)

Somewhat inadvertently, Aspnes had created a more egalitarian community, and a new kind of learning environment. It's a remarkable fact that at any given moment there are tens of thousands of people doing creative writing and computer programming in their spare time just for fun on MUDs.

The rest of this section describes what a MUD is by analyzing its three primary components: personas, rooms, and objects.

### 1.3.2 Personas

When you first connect to a MUD, you must pick a name. You may also chose your gender and describe what you look like. If we are in the same room on MOOSE Crossing and you type "look at Amy," you will see:

```
Amy is 5'8 with shoulder-length wavy brown hair, and a mischievous
grin which seems to say "Can you believe I get paid to do this?"
She is awake and looks alert.
Carrying:
  randomizer                               Generic Returning Object
  A Carrot Cake                            Amy's card
  generic business card                    frob
```

On some MUDs (including MOOSE Crossing), participants may have more than one persona and may switch between them, as Hermes demonstrated in Section 1.1 by becoming Cloudstreamer and Widget, and Miranda demonstrated by becoming Athena. Each persona has a different description and gender. Rachael (girl, age 12) extended the software on MOOSE Crossing so each persona can also have different clothes. MUDders often work through issues of personal identity through the activity of constructing and role playing different personas (Bruckman 1992; Turkle 1995).

Sometimes MUD players are explicitly taking on fictional roles, pretending to be someone else; however, at most communities most of the time, participants are primarily just being themselves—it's more like a costume party than an actor's workshop. On MOOSE Crossing, only small amounts of real role playing take place. Notably, Rachael organized a sub-community she calls "The Nest" for medieval role playing. Nest members have homes at The Nest Gathering Grounds, and special nest personas who pretend to be living in medieval times and not to know about things like telephones and computers. Different MUD communities vary in what percentage of the time players typically are playing a role versus simply being themselves. People tend to stay "in character" more of the time on MUDs with themes taken from popular culture such as Star Trek or Anne McCaffrey's *Dragonriders of Pern* series of books. However, on most MUDs most of the time, people may have fanciful names and descriptions, but they are primarily talking about issues of real-world concern to them.

This is certainly true on MOOSE Crossing. For example, Miranda's alter-ego SuperMiranda talks in all capital letters and jokes about being a super hero. However, she usually changes into this persona very briefly, and then changes back. This is the only one of her several personas which behaves at all different from the others. Whichever name she is using, she is primarily just being herself. She talks to the other children on MOOSE Crossing about the things she's making there, and about elements of her real life (her new braces, her friends, the first day of the new school year). Similarly, Hermes, Cloudstreamer, and Widget don't behave observably different. Jo (girl, age 12-13) tries to behave differently when she is her gothic alter-ego Rowena, but she doesn't entirely succeed. Here is Rowena's description:

```
You see a tall girl of about 16. Her hair is died black and she is
wearing a long black dress. Her complexion is very light, almost
white, as if she has never seen the light of day. Her face never
smiles, though she's really quite friendly.
```

In one conversation, she talks about how successful this persona is with Scribbles and Werdna (boy, age 9). Werdna is in another room, but is talking to the others through his puppet "hawk":

```
Rowena says, 'scrib, on a scale of one to 10, 10 being a total goth,
   how gothic is my character?  You too werdna'
hawk [Werdna] says, '8 and a half'
Rowena says, 'better than I expected, but what's missing'
Scribbles says, 'your character looks gothic, but she doesn't act
   like it.  She acts like my Stacey:)'
[Stacey is Rowena's real name.]
hawk [Werdna] says, 'well i couldn't say'
Scribbles says, 'and htat would sound odd to the outsider...'
Scribbles says, 'but that's ok.'
Rowena says, 'I'm still trying to get into it, its hard being
   depressed when your pale purple'
```

Despite the fact that the children rarely explicitly role play, the way a child (or adult) chooses to describe himself or herself is still a window onto that child's sense of self. It's not an accident that Miranda's younger sister chose the character name "Mouse." As the younger child in the family, Mouse (girl, age 8-10) has an acute sense of being small and not able to do all the things her big sister can do. Her self confidence (and her writing ability) have grown over the last year. Contrast the description she made the day she joined MOOSE Crossing to one made more recently:

```
a blown up, purple mouse.  She has a weel on her butt.  [November 1995]
```

```
A little baby tulip. She is Red and very cute. She has blue eyes and
a small mouth. She is the cutest tulip you've ever seen.  [September
1996]
```

The earlier description conveys a sense of being small, unattractive, and trapped. (How would a mouse get around with a wheel stuck to its behind?)[4] The more recent description in contrast is attractive and confident. She's still small, but she's proud of being small, and feels good about herself. Describing a fictional persona provides people with an unusual opportunity to express how they feel about themselves, and potentially to reflect on that expression.

### 1.3.3 Places

The virtual world of MUDs is structured into different "rooms." When you "say" something, it is heard by others in the same room. If I type:

```
--> say hi
```

I see:

```
You say, 'hi'
```

[4]In November 1996, I asked Mouse to describe her very first MOOSE Crossing description (written a year before), and she remembered it clearly. Asked why she chose that particular description, she laughed and said she had *no* idea. She said she had imagined the wheel not as being something the mouse could ride on but as something just stuck there. It was not the sort of wheel a mouse runs on, but a regular wheel.

Everyone else in the room sees:

```
   Amy says, ‘hi’
```

You can also “emote” to take actions. If I type:

```
   --> emote laughs
```

Everyone in the room (including me) sees:

```
   Amy laughs
```

If you want to talk to someone not in the same room, you can “page” them. Hermes began his conversation with Miranda in Section 1.1 by paging her asking if it was OK for him to come over. It’s considered good manners to page someone before barging into their room.

Places provide contexts for social interaction. On MediaMOO (see Section 1.4), people behave differently if they are in my office than if they are in a café or the ballroom. On MOOSE Crossing, kids behave differently if they are swimming at Miranda’s Paradise Island or Jack’s swimming pool than if they are standing in the library. Here’s a transcript of Rufus and Hermes going for a swim in Jack’s pool. Jack is a thirteen-year-old boy who is not connected at the time of this log, but his pool is still there for others to enjoy. Jack has programmed the pool so that you can do lots of things there including dive, surface, splash someone, dunk someone, and swim laps. Rufus drives Hermes there in Sparky II, his bus with homes in the back:

```
   Rufus says, 'We are here!!'
   Rufus says, 'Just type exit.'
```

As they each get out of the bus, they see:

```
   You are in a nice pool,  with clear blue water.  YOu can set the
   tempiture of the pool, and you can dive and swim under water.  You
   can also dunk peoples, splash, lap, check underwater, talk
   underwater, and surface from underwater!
   You see fluffey and Sparky II here.
   Austina [*sleeping*], Alexander [*sleeping*], and Jack [*sleeping*]
   are here.
```

Hermes changes into his dolphin personality, Flipper. Rufus says, 'Watch this!' and types “dive.” Jack’s dive program prompts him “Describe your dive. Put it in the form of an emote,” and then asks him for the name of the dive. It puts these two things together and prints the performance of the dive to everyone in the room:

```
    Rufus climbs up onto the diveing board and takes the form of a
      Rufusflop.  Then, Rufus flips off the board and emote flips into
      the air five and a half times before doing a bellyflop.!
```

Jack's program's instructions are a little confusing—Rufus didn't need to proceed his description of what he does with the word "emote." Hermes/Flipper types "splash Rufus," but Jack's program tells him "you can't splash someone who is below the surface of the water..." Jack's pool keeps track of who is underwater, at the surface of the water, and by the side of the pool. Rufus types "surface" and they both see, "Rufus comes back out of the water!" Flipper tries "splash Rufus" again, and this time it works:

```
Flipper splashs Rufus!  Rufus is all wet!
Rufus says, 'How was it?'
Rufus says, 'Hey!'
```

Rufus types, "dunk Flipper" in revenge for being splashed. They see:

```
Rufus dunks Flipper!  Flipper comes up sputtering!
Flipper says, 'a 10'
Rufus says, 'Thanks. Go ahead and try it!'
Flipper says, 'ok'
```

Rufus looks at Flipper. He sees:

```
An 8 foot long bottle nosed dolphin with a mischievous grin. He sees
you looking at him and splashes you with his powerful tail. You are
thoroughly drenched!!! Flipper winks at you and then dries you off
with a mighty blast of air from his blowhole.
He is awake and looks alert.
Flipper is wearing a moose-crossing uniform.
```

Flipper types "dive" and describes his dive to Jack's program. They see:

```
Flipper climbs up onto the diveing board and takes the form of a
   tripl tail dance.  Then, Flipper flips off the board and :erupts
   from the water and does a tail dance 3 times!
Rufus says, 'WOW!! A 25'
```

MUDs are more than chat rooms. The metaphor of being at a pool gives Rufus and Hermes/Flipper a context for creative play. The pool is a project completed by another child, Jack. Many children have told Jack how much they like his pool, and this provides him with positive reinforcement. The MOOSE programs that make up Jack's pool are available for the other children to use as coding examples. Furthermore, the simple knowledge that a peer made something like the pool can help inspire other children to attempt significant projects of their own.

### 1.3.4 Objects

In addition to people and places, MUDs are filled with objects. On MOOSE Crossing, the most popular kind of object are pets. It's possible to make pets follow you through the virtual world. Most members have at least one pet following them, and some have several. For example, Cindy (girl, age 10) was initially followed by quite a noisy menagerie. Here are the messages generated by her arrival in a room:

```
Hoover waddles in, following after you.
Kanga arrives, following after you.
Sandy arrives, following after you.
Peggy arrives, following after you.
Sandy sniffs Peggy curiously.
Peggy says, 'HiKanga'
Peggy says, 'HiSandy'
Peggy says, 'GreetingsHoover'
```

Cindy would later leave some of her pets in her home, but for a month after she made them she had them all follow her. MOOSE (and the MOO software on which it is based) are "object-oriented" (see Chapter 3). Hoover is a penguin. Cindy made this by making an object which inherits from generic penguin, a program that Rachael wrote. It immediately had all the abilities Rachael designed for it, including reacting differently to six kinds of food and automatically keeping track of how hungry it is based on how long it's been since it last ate. Later, Cindy would add six new programs to Hoover, making it possible for her to kiss it, hug it, dress it up, make it greet people, and make it eat additional kinds of food. Examining any object gives you a list of commands that work with that object, so you don't need to guess. One of the scripts Rachael wrote on generic penguin lets you feed it shrimp. Here's an example:

```
--> feed Hoover shrimp
You can't feed Hoover shrimp today because Hoover is on a diet.

--> diet Hoover
Hoover is no longer on a diet!!!

--> feed Hoover shrimp
You feed Hoover a ton of shrimp!
Hoover gobbles the shrimp down in seconds.
```

Here is what five of the programs Cindy added to Hoover do:

```
--> hug Hoover
Hoover snuggles into your arms

--> kiss Hoover
Hoover kisses your nose tenderly

--> feed Hoover green eggs
```

```
Hoover jumps a few feet in the air flapping his wings franticly, then
   he calms down by bowing politely to you

--> undress Hoover
Hoover blushes bright red.

--> hi Hoover
Hoover bows shyly, and then tries to hide in Cindy's pants
```

Kanga is a kangaroo. Cindy started making Kanga by doing a tutorial which helps you to make a dog and changing it to be more kangaroo-like. She then added several new abilities not in the tutorial, including making it ticklish, and making it eat daisies. Peggy is a pet pig that Cindy bought at Byron’s Pet Shop. When you buy a pig, Byron’s storekeeper will make you an object which inherits from his generic greeting creature object. Greeting creatures say hello to every object which comes into the room. As Cindy wanders around the world, Peggy says hello to Cindy’s other pets as they arrive in the room, following Cindy! After she bought Peggy, Cindy added five new programs to her. She decided that Peggy wasn’t really a pig but a piggy bank, and wrote a program to let you feed Peggy pennies. Cindy made Sandy by making an object which inherits from generic smart dog, a program that I wrote. She then added seven new programs to Sandy to make it do other dog-like things. The programs on the parent object make Sandy sniff new things that come into the room. Sandy sniffs Peggy in the transcript above because Peggy arrived in the room after Sandy did.

Objects can serve a variety of functions. They can perform useful functions like the research directory on MediaMOO, or the object transfer station on many MOOs and on MOOSE Crossing which lets you change who owns an object. They can also be conversation starters. In real life, one doesn’t normally talk to strangers on the street. However, there is an unwritten rule in many cultures that it’s acceptable to begin talking to a stranger who has a dog. Admiring the dog and asking questions about it opens the door to conversation on other topics. The same happens in the virtual world with virtual dogs, and other kinds of objects.

Making objects gives individuals opportunities for creative expression. The ways in which making things together helps to strengthen a community are discussed in detail in Chapter 6, “Constructionist Culture.” Finally, objects can be as much an expression of self as personas. Arjun Appadurai writes about how much you can tell about a family by the objects in their home (Appadurai 1986). The same is true in the virtual world. You can tell a lot about a person by the things they chose to buy in the real world, and even more by the things they chose to make in the virtual world. Creating objects provides opportunities for self expression, creative play, and learning through design.

## 1.4 Prior Work: MediaMOO

MOOSE Crossing is the second virtual community I have designed/founded. The first, MediaMOO, is a MUD designed to be a professional community for media researchers. MediaMOO opened to the public in January 1993, and as of October 1996 had 1000 members from 35 countries. From the start, MediaMOO was intended as a kind of rehearsal for MOOSE Crossing. It was my first experience applying constructionist ideas to the design of a virtual community. Throughout this thesis, I will occasionally refer to MediaMOO and design principles learned through the MediaMOO project. I will very briefly introduce that project in this section.

Most MUDs are populated by random collections of people with little in common. As a result, the conversation often sinks down to the least common denominator of discourse. On MediaMOO, I decided to see what would happen if I brought together a community of people with shared interests—in this case, the design of new media technologies. To become a member of MediaMOO, you must be doing some sort of “media research.” In evaluating applications, I try to be loose on the definition of media and more strict about the definition of research. Everyone on MediaMOO “wears” a description of his or her research interests, and is identified by his or her real name and email address. It’s rather like an endless reception for a conference on media studies.

In the initial design of MediaMOO, I tried to maximize opportunities for individuals to extend the virtual world. On most MUDs, you must ask to be allowed to program new objects. On MediaMOO, everyone automatically has that privilege. I constructed very little of the virtual world, but instead tried to maximize opportunities for people to build their own spaces. The world that has emerged reflects the rich diversity of its inhabitants. A more detailed discussion of MediaMOO appears in “The MediaMOO Project: Constructionism and Professional Community” (Bruckman and Resnick 1995).

## 1.5 The MOOSE Crossing Project

The MOOSE Crossing Project has three major components: the development of new technologies, the use of that technology with children (including a weekly after-school program at the Media Lab), and the analysis of the children’s activities and learning.

- **Tools**

If kids are to work on significant projects in a MUD, they need access to quality tools designed specifically for their needs. The MOOSE Crossing project began with the realization that they would need both a new programming language and a new client interface. Work on the design and development of the MacMOOSE client began in September 1992. Work on the MOOSE language began in September 1993. Previous MUD languages were designed for adults; MOOSE is the first to be designed explicitly for children. The client and language were first used by children in October 1995. Chapter 3, "Designing Enabling Technologies: The MOOSE Language and the MacMOOSE Client," describes the design of these new technologies in detail.

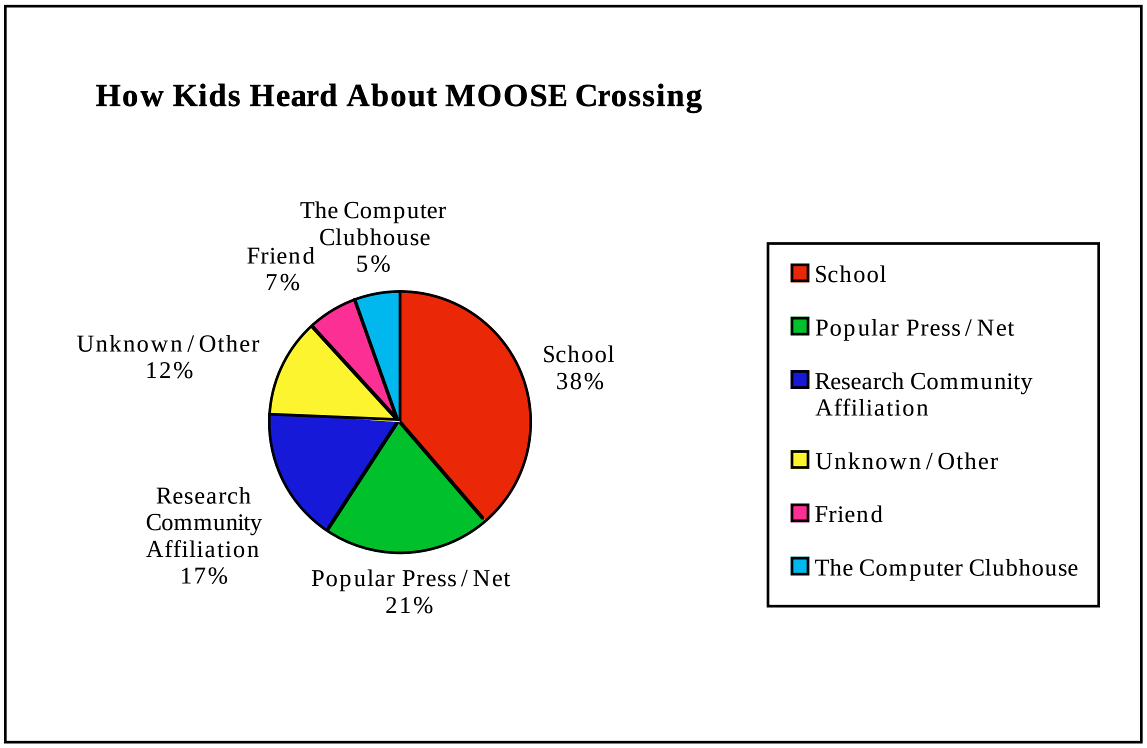


Figure 1.1: How Kids Heard About MOOSE Crossing

- **Use By Kids**

MOOSE Crossing is used by children from home, in after-school programs, and as part of regular school activities. From October 1995 through February 1997, a group of approximately six children came to The Media Lab once per week to participate in a regular MOOSE Crossing after-school program. The main purpose of the program was to give me the opportunity to study a group of children using MOOSE Crossing closely. Exact membership in the group changed occasionally (particularly over the summer), but remained largely the same over the course of the year. Each session lasted approximately two hours. While we originally intended for MOOSE Crossing to be used by kids nine years of age and older, several as young as seven have participated successfully. As of March 1997, over 160 children had participated. While home users form the largest group, the fastest-growing group is children using MOOSE Crossing from in-school programs. Figure 1.1 shows how kids heard about MOOSE Crossing, and Figure 1.2 shows where they connect to MOOSE Crossing from. This data is compiled from the questionnaire participants complete on registering. Some children start off participating from school or an after-school program and add additional home access later. The data in Figure 1.2 represents their initial answer to the question “From where do you plan to connect to MOOSE Crossing? (home, school, community center, friend's house, or something else).”

The population is more gender-balanced than many other communities on the Internet. Boys make up 56% of the population, and girls are 43%. See Section 7.1.3 for more on gender issues.

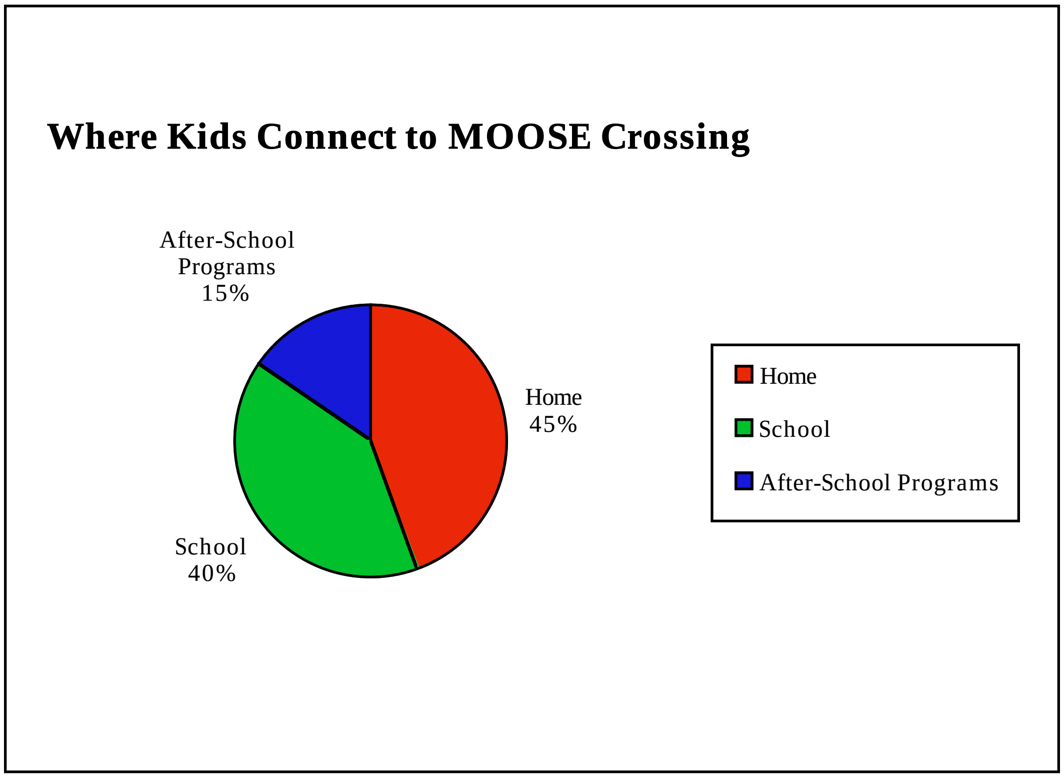


Figure 1.2: Where Kids Connect to MOOSE Crossing

- **Analysis of Children’s Activities and Learning**

The after-school program at the Media Lab gave me the opportunity to get to know the children and follow their progress closely, using an ethnographic methodology. After each session, I took detailed field notes. I also interviewed them periodically on video, using a clinical interview style.

Everything typed on MOOSE Crossing is recorded, with written informed consent from the parents and children. This has given me an opportunity to study individual children’s learning experiences in detail. The total amount of data recorded is staggering—over 700 Mb as of March, 1997. It’s impossible for one person to review even a small fraction of it; however, it is possible to study selected individuals, places, or objects in great detail.

Recording this data is somewhat of an invasion of the children's privacy. Privacy rights are an issue I'm concerned about (and volunteer some of my spare time lobbying for), and this was not a decision I took lightly. However, after giving the issue serious consideration, I decided that the benefits outweighed the problems. MIT's Committee on the Use of Humans as Experimental Subjects (COUHES) subjected the plans for MOOSE Crossing to careful review before approval. Every MOOSE Crossing member signs a permission/code of conduct form on paper and mails or faxes it to MIT before they are allowed to participate. Minors must also have their parents sign a parental permission form. These forms explain what data is being recorded and why. The formal nature of this process makes me feel more comfortable that the consent being obtained is informed consent—members didn't just click "OK" to a screen of fine print, but signed a statement on paper that spells things out. Children are unlikely to understand the significance of this fact, but their parents are empowered to make this decision on their behalf. It's unclear to what extent the existence of log files has changed the social atmosphere on MOOSE Crossing.

In writing about the children's experiences, I chose to change their character names and names of some of their objects and pets. Some people believe that since MUD names are already pseudonyms, no further disguise is necessary. I disagree. People invest a great deal in their virtual identities. Even if those are not directly traceable to their real identities, the virtual persona itself merits protection. Unfortunately, in a community as small as MOOSE Crossing, it's impossible to disguise individuals in such a way that insiders won't recognize them. Many researchers give their subjects added protection by hiding the community name, but that was clearly not possible in this case. As a result, since the subjects might be identified by some readers, I have edited out unnecessary details in a few unusual places where I am concerned those details might embarrass the child in question. For example, information about a particular child's difficulties getting along with his or her parents or siblings might indirectly shed some light on that child's learning experiences; however, that information is not essential. The focus of this work—on learning, construction activities, and community—has made it possible to eliminate such details without damaging the intellectual integrity of the thesis. A more sensitive topic (such as the work I did with Sherry Turkle on conceptions of the self in MUDs) would be impossible to study in this situation and still protect the subjects adequately.

## 1.6 Outline

This thesis is divided into seven chapters. It was designed to be readable in any order; consequently, some basic facts are repeated in multiple places. This first chapter serves as an introduction to the project and the motivations behind it. Chapter 2, “The Day After Net Day: Educational Approaches to Using the Net” reviews educational approaches to using the Internet. Applications which focus on using the net to deliver, retrieve, or share information are separated from those which see the network as a place to create a “technological samba school.” Chapter 3, “Designing Enabling Technologies: The MOOSE Language and the MacMOOSE Client,” introduces the new technologies designed as part of this project, and examines the ideas underlying their design. Chapter 4, “Collaborative Learning Strategies: Storm’s Weekend with Rachael,” is a “thick description” (Geertz 1973) of a weekend on which one twelve-year-old girl taught another twelve-year-old girl how to program. It analyzes the learning strategies children typically use on MOOSE Crossing. Chapter 5, “Community Support for Construction,” examines in detail the ways that the community motivates and supports individual children’s learning experiences. Chapter 6, “Constructionist Culture,” examines the converse—the ways in which building things together helps to create a particularly felicitous kind of community. Chapter 7 presents a number of open questions for further research, and then examines what conclusions can be drawn from this work about construction, community, computers, and learning. The Appendix lists all the programs and descriptions of six randomly-selected children who participated in MOOSE Crossing in a variety of settings.

# 2. The Day After Net Day: Approaches to Educational Use of the Internet

## 2.1 The Hype and the Reality

On Saturday, March 9th, 1996, volunteers filled California schools to wire them for Internet access. As many as 150 volunteers showed up at some schools. It was a high-visibility event—even the president and vice president of the United States joined in: "Donning electrician's gloves and hopping on a ladder, President Clinton joined the cyberspace revolution Saturday as he worked with Vice President Al Gore to install about 70 feet of pink, white and blue conduit at a Contra Costa County high school," wrote the *San Jose Mercury News* (1996). The organizers of the event, dubbed "Net Day", reported that over 18,000 volunteers participated.

The day after Net Day, teachers were left with questions: now what? What exactly are we supposed to do in our classrooms with this new technology? Contrast the utopian hype surrounding Net Day to this letter to the editor published in *The Boston Globe* a few months later:

> "Massachusetts schools should consider themselves fortunate to be in 48th place ("A Net gain for schools," editorial, May 28). Having just spent more than two frustrating weeks trying to get on and use the Net, I can assure teachers that it is one of the greatest wastes of time ever foistered upon the public. Not only is it hard to find the place you're looking for, but when you finally get there the information you hoped to find is not available or of limited value. The main purpose seems to be to amuse browsers who have unlimited time with sluggishly transmitted, cute pictures and endless alternatives to "click on." The only benefactors from wiring up the schools will be equipment sellers, installers, and the inevitable service providers." (Kleinschmidt 1996)

The positive and negative hype are equally comic. The letter's author has little idea how one might use the Internet in an educational setting. However, in a sense, no one does—the possibilities are still being explored. In the popular press and the popular imagination, The Net functions largely as a symbol.[1] In the positive hype: "The Net is the future. The Net is progress. If your child is using The Net, then he or she is part of the future; your child will be a success." In the negative hype: "The Net is Technology. Technology has cheated us before and is trying to cheat us again. Technology will bring us no real benefits. The Net is not just a waste of time and resources—it is diverting us from the core values that really matter." In the past, other technologies have played this symbolic role. In the 1980s, computers in general tended to symbolize the future; in the mid-1990s, people

[1]See (Arnold 1995) for a discussion of ways in which the Logo programming language functions as a symbol for a particular progressive educational tradition.

are more likely to use the Net as that symbol. The role of symbolizing the future is constantly migrating to a newer technology. If The Net functions as a symbol, Children function as an even more powerful symbol: “Children are the future. Children are innocent, pure, and impressionable.” The combination of these two symbols, Children using The Net, is a cultural powder keg. When people debate the issue they are often really debating their hopes and fears for the future—their personal future as well as the future of our society.[2] The reality, the real things people are doing in classrooms with children and net connections, is much more pedestrian.

**EDUCATIONAL APPROACHES TO USING THE NET**

| | | |
|---|---|---|
| **I.** | **Distance Education** | |
| | *Tradition:* | *Examples:* |
| | Instructionism | The Open University |
| | | IBM in China |
| | | Diversity University |
| **II.** | **Information Retrieval** | |
| | *Tradition:* | *Examples:* |
| | Exploratory Learning | Net surfing |
| | | Research projects |
| **III.** | **Knowledge-Building Communities** | |
| | *Tradition:* | *Examples:* |
| | Collaborative Learning | Global Science |
| | | CSILE |
| | | Professional Communities |
| | | Computers & Writing |
| **IV.** | **Technological Samba School** | |
| | *Tradition:* | *Examples:* |
| | Constructionism | MicroMUSE |
| | | Pueblo |
| | | MOOSE Crossing |

Table 2.1: Educational Approaches to Using the Net

One common mistake is to think of The Net as one thing. Students and educators use computer networks in a wide variety of ways. Each approach is rooted in different educational traditions. Broadly speaking, you can put

[2]I recently met with an educator from a third-world nation, and invited her to arrange for some children from her country to join MOOSE Crossing. The coordinator of her visit sent me a very nice thank-you note which commented that “She was especially interested in the “virtual community” project, in which our schools can participate in the future.”

educational uses of the net in four categories: distance education, information retrieval, knowledge-building communities, and technological samba schools (See Table 2.1). In the rest of this chapter, I'll discuss each approach in turn. As we move from approach I to IV, the emphasis shifts from information to ways of knowing, and there is an increasing emphasis on community. There is also a shift from more curriculum-centered approaches to student-centered approaches. The particular projects selected for discussion were chosen to highlight different pedagogical approaches. The list is far from comprehensive.

## 2.2 Distance Education

Long before computers were invented, people were learning from home via correspondence courses. For the geographically isolated and for adult learners juggling the demands of work and family, distance education has provided otherwise impossible opportunities. Britain's Open University currently serves the needs of 200,000 students. Their web page (*http://www.open.ac.uk/*) notes that "The oldest graduate so far was 93, while the youngest student is a nine-year-old prodigy taking maths. There are roughly equal numbers of men and women. About three-quarters of students remain in full-time employment throughout their studies."

The tradition of distance education is rooted in "instructionism." An instructionist approach to education focuses on the transmission of information from teachers to students. Students are expected to master a curriculum-specified set of facts, and be able to repeat those facts on examination. Mastery of information is emphasized over ways of thinking and knowing. Most commonly, distance education students receive a set of materials to study, and then take tests to demonstrate their mastery of that information.

Clearly missing from this model is classroom interaction. The Open University has tried to counteract this problem by setting up local networks of tutors and regional centers around the United Kingdom. They are currently beginning a major initiative to use computer networks to provide access to information and enhance interaction among students and teachers.

Many distance education projects are experimenting with video conferencing techniques. An expert's live presentation can be sent to thousands of students. Students can ask questions from remote locations. Questions and answers can be broadcast to all students participating. Proponents argue that students who would normally have access to only inexperienced teachers now are being taught by world-class experts. Underlying this argument I believe is a fundamental misunderstanding of what it means to be a good teacher, and a lack of respect for the teaching profession. Consider the combinatorics of the situation: if one expert is lecturing to hundreds or thousands of students, there will be time for only a tiny percentage of those students to ask questions. The entire presentation is then equivalent to students learning from watching a videotape. How would one compare learning from videotapes of experts to learning from a good teacher? Ideally, the teacher establishes a relationship with each student, getting to know him or her individually. The social and psychological dimensions of those relationships are as important as any role the teacher may play in supplying information or assessing students' performance. The teacher tailors the learning experience to meet student's needs, rather than being tied to a fixed curriculum. The teacher's role varies in different pedagogical traditions, but across all of those traditions one thing is clear: good teaching is an art.

In early 1996 I met with a development team from IBM who were working on just this sort of solution for China—piping video into classrooms across computer networks. They argued that the quality of teaching in China is generally horrible and the number of learners so immense that this sort of network was an appropriate solution. This is not only a waste of scarce financial resources, but also could be actively harmful to the educational process if teachers perceive the lack of respect for their skills and their efforts that motivated this system design. Instead of dismissing those teachers as incompetent, why not invest resources in teacher training and professional development? That would bring more benefit to students than talking-head video presentations. (See Section 2.4.3 for more on supporting teachers.)

A somewhat better use of technology to support distance education involves the use of mailing lists, real-time chat, and MUDs to foster interaction among students and teachers on a reasonable scale. Since these technologies are many-to-many instead of one-to-many, they afford more real interaction. For students taking classes at The Open University, these technologies are providing new opportunities for students to learn from one another.

At a MUD called Diversity University (*telnet://erau.db.erau.edu:8888*), students sit at virtual desks in virtual classrooms. The designers have tried to move the classroom environment into text-based virtual reality, complete with programs to simulate white boards and white-board erasers. Since the nonverbal cues that help people negotiate whose turn it is to talk are absent, many classrooms include software to programmatically control turn-taking. While this approach is certainly preferable to talking-heads videos, it is still far from ideal.

Distance education often uncritically gives us a bandwidth-impoverished literal-minded copy of the traditional classroom. In most of these projects, the metaphor of having a virtual space is being taken too literally. Virtual classrooms are not simply mediated forms of real classrooms. To treat them as such is akin to early filmmaker who pointed cameras at theatre stages and produced essentially filmed plays. Virtual spaces are a new medium whose properties need to be explored and used to their best advantage. More ambitiously, this new technology can be used not merely to reproduce traditional education, but to help reform it. New educational technologies can provide opportunities to introduce new educational ideas. Most distance education projects simply translate an old medium (the classroom) into a new one (virtual space) without reflecting on either what the new medium is good for or how the old medium needs to be reformed.

## 2.3 Information Retrieval

When the general public think about children using the Internet in school, they most commonly think, as did the author of the letter to *The Boston Globe* (quoted in Section 2.1), that the children will be "surfing" the net for information. From this perspective, teaching children about the Internet is the modern equivalent of classes in library skills. Learning how to find information online is a useful means to an end—not an end in itself.

Using the Internet as an electronic library has a number of pedagogical benefits when used in combination with (not instead of) other information sources. The volume of information available exceeds that possible within a school library, and much of that information is more current than is possible in printed books. It's significant that on the Internet, all schools—rural and urban, rich and poor—gain access to the same quantity and quality of information (except where filters are imposed to protect the children from controversial information.) However, it is not clear that it's of central importance for students to have access to the latest information; most school libraries are more than adequate for students' needs. On the other hand, the idea that they have access to the latest information has the potential to get kids more excited about what they are researching. Students often feel condescended to by schools and school text books. By giving them access to "real" information sources used by adults, they can be made to feel that they are being taken seriously, and they may consequently take the educational process more seriously themselves.

Many express concern that much of the information available online is not accurate. While this is a problem, it also has a hidden benefit. Children are taught not to believe everything they hear, but they are not urged strongly enough not to believe everything they read. The network brings issues of point of view and reliability into high relief. It's likely that children raised using electronic information sources will learn to be more critical consumers of all information.

At its best, information retrieval activities are a form of exploratory learning: children research a topic they care about in depth. They evaluate the information they discover critically. The research culminates in a report or other project. At its worst, information retrieval can become a kind of trivial pursuit game. In a classroom I once visited, students were challenged to find the names and dates of the largest volcanic eruptions in history. This wasn't presented as part of a larger curriculum unit on volcanoes—the information was not situated in any meaningful context. It was merely an academic scavenger hunt. Thoughtful uses of electronic information retrieval in the classroom have more to do with traditional research projects than with "net surfing."

### 2.3.1 Children Accessing Controversial Information

A complicated and troubling issue raised by this technology concerns children's potential access to controversial information online. A tremendous volume of obscene, racist, and violent information is available online. While such information generally appears only when one actively looks for it, it is possible to stumble on it accidentally, as one of my students did several months ago. One of MOOSE Crossing's first sample programs was an elephant that tells elephant jokes. A twelve-year-old boy using MOOSE Crossing at The Media Lab wanted to make a lawyer who tells lawyers jokes. He asked one of the adults present if it was OK for him to open up a web browser to search for lawyer jokes. I was working with children on the other side of the room, and heard about this a few minutes later. Something troubled me about it, but I wasn't immediately sure what. I was surrounded by kids demanding my attention, and didn't stop to give the matter my full attention right away. Ten minutes later when things had quieted down, it occurred to me: the last time I saw a joke collection posted to the web, many of the jokes were dirty ones. Looking over the student's shoulder, my fears were proved correct—the joke collection he was reading was largely obscene. I had a talk with him about the responsibilities that come with the privilege of net access, and the reasons why many people don't want their children to see such material. Whether any real harm was done depends on your perspective.

I've told this story to a number of adults who have chuckled and laughed it off—there's no real harm in a dirty joke or two, is there? There are two problems with this argument. First, not all parents agree. Some would find the fact that their child had been exposed to a dirty joke to be a very serious matter. Second, the level of obscenity in some materials available on the net exceeds what you might guess—the lawyer joke collection in question included anal sex jokes. While I believe our culture would benefit from more open discussion of human sexuality, the fact remains that such subjects should be broached at an appropriate time and in an appropriate context. A student actively seeking such information should be able to find it, but he or she shouldn't stumble across it while looking for lawyer jokes.

There are a wide variety of strongly-held opinions on this issue. In March of 1995, Michele Evard and I founded an email discussion list on Children Accessing Controversial Information (caci). We led a round-table discussion on the topic at the April 1995 meeting of The American Educational Research Association (AERA). On the caci list, the topic was sufficiently controversial to generate a high volume of messages, and a high level of emotional tension. Many participants presented their views with absolute certainty, refusing to acknowledge the merit of other people's points of view. On one side, some people argued that freedom of speech and the freedom to read are absolute, and any restrictions are a violation of basic human rights. On the other side, some people declared that children finding inappropriate information online would be nothing less than tragic, and must be prevented at all costs. After several months, we found a volunteer to take over the list management, and I unsubscribed from the list. The repetitive nature of the conversation and its self-righteous tone were more than I could stomach. Many people feel strongly about this issue.

In March of 1996, I organized a session at MIT's Communications Forum entitled "Protecting Children / Protecting Intellectual Freedom Online." Judith Krug, Director of the Office for Intellectual Freedom at The American Library Association, and Bill Duvall, President of SurfWatch Software, spoke. Krug spoke eloquently about the importance of intellectual freedom. However, she failed adequately to acknowledge that libraries do make editorial decisions in what books they chose to buy. Duvall presented his company's SurfWatch software, which gives parents the option to filter out controversial information. SurfWatch uses a combination of keyword filtering and ratings by human reviewers to filter out sexually explicit content. It does not filter violent or racist content, and parents can in no way tune the software to match their values. While Duvall made a good case for the value of empowering parents to make choices for their children, he failed to address the issue of the competing rights of children, parents, school systems, and the broader society. For example, should teenagers be able to access information about gay teen support groups if their parents and school system don't want them to? Does a local school district have the right to ban access to information about evolution if the broader society believes it to be an important scientific fact? None of these issues are new; the Internet simply gives them a new immediacy.

SurfWatch is only one of a growing number of products designed to make net access more appropriate for kids. There's also Cyber Patrol, Net Nanny, SafeSurf, and others. Perhaps the most intriguing is the Platform for Internet Content Selection (PICS), a research project aiming to design a platform to support not only multiple ratings of content but multiple ratings systems (Resnick and Miller 1996). While PICS has promise, all attempts at technological solutions to fundamentally social problems have limitations. The most useful analogy I've come across is that the Internet is a city. You don't let a very young children go into the city alone, but you might let them play alone in the yard. (In this scenario, MOOSE Crossing is like a loosely supervised playground.) As children grow older, you need to educate them on how to be street smart. Eventually, you need to trust them to venture off on their own and use good judgment. The Internet brings some of the complexities of the real world onto your desktop. Parents need to stay involved to help children learn to negotiate those complexities.

## 2.4 Knowledge-Building Communities

Distance education focuses primarily on information delivery. Information retrieval is in a sense the opposite process. Knowledge-building communities focus on information *sharing*. Distance education is probably most closely allied with the behaviorism of B.F. Skinner (Skinner 1968) and information retrieval at its best is allied with the exploratory learning of John Dewey (Dewey 1938), knowledge-building communities are more closely linked to the work of Lev Vygotsky (Vygotsky 1978). Vygotsky emphasized the social, collaborative nature of learning.

### 2.4.1 Global Science

The most common kind of knowledge-building community is what I have dubbed "global science" (See Table 2.2). In the TERC/National Geographic Acid Rain Project (now part of the Global Lab Project), children around the world record data about acid rain in their area. By sharing the data, they are able to gain an understanding of this global phenomenon. Similarly, in The Journey North, children from Mexico to Canada collaborate to track animal migration patterns. In the Kids as Global Scientists project, children collaborate to study weather patterns (Songer 1996). The Jason Project has a slightly different emphasis. In the Jason Project, marine biologists from Woods Hole Oceanographic Institute explore undersea phenomena with a small remote-controlled submarine. Children can see video output by the sub, and take turns controlling it remotely. The goal is for the children to develop a sense of having participated in a "real" scientific investigation. These are just a few of the many "global science" projects on the Internet.

As part of the Kids as Global Scientists Project, Songer compared learning outcomes for a class of students discussing weather data with other schools over the Internet, and a control group doing research in their school library. While the quality of scientific data collected was comparable between the two groups, "Internet responses focused on a mixture of scientific and personal information, such as personal anecdotes or familiar occurrences in local weather patterns." The Internet students had become more personally involved with the project. Teachers observed that their Internet students had increased motivation to learn about the weather (Songer 1996).

**"GLOBAL SCIENCE" PROJECTS**

- TERC / National Geographic Acid Rain Project (Global Lab)
  *http://hub.terc.edu/terc/gl/global-lab.html*

- The Jason Project
  *http://www.jasonproject.org/*

- The Journey North
  *http://www.ties.k12.mn.us/~jnorth/index.html*

- Kids as Global Scientists
  *http://www-kgs.colorado.edu/*

- The Noon Observation Project
  *http://www.ed.uiuc.edu/courses/satex/sp96/noon-project/index.html*

Table 2.2: "Global Science" Projects

In The Noon Observation Project, students repeat an experiment first performed by Erasthones in Ancient Greece: measure the length of a shadow at noon at several distant locations, and you can estimate the circumference of the earth. The study organizers write:

> In order to learn about the role that the network played in this project, let us consider whether such a project could have been done without an electronic network. In terms of conducting a project which provides a practical context for mathematics skills, the class could have gone out and used their meter stick shadows and the shadow of the school's flagpole to determine the height of the flagpole, as mathematics teachers have done for generations. However, the network seemed to provide a highly motivating context for learning, both for the students and for the teachers involved. More specifically, it provided support in the following ways: 1) as a source of ideas, 2) as a supplier of tools, 3) as a source of diverse data, and 4) as a diverse audience (Levin, Rogers et al. 1989).

Perhaps the most important of these criteria is the notion of audience. In The Instructional Design Software Project, Idit Harel had fourth grade students write software to teach third graders about fractions (Harel 1991). Harel notes that the educational benefit goes entirely to the fourth graders—the third graders learn little if anything from the experience. However, the notion of having an audience restructures the fourth graders' relationship to the design process. Students become more excited about the project and take pride in the quality of their work, because they are designing for an audience they care about. The same phenomenon can be observed in global science research projects: the idea of sharing their data with an audience of their peers is motivating, and encourages students to do quality research. Given the often limited nature of the interaction among students in these projects, it is often the idea of having an audience that is beneficial, more than the actual interaction with that audience that takes place.

### 2.4.2 CSILE

In a knowledge-building community, ideally students should critically debate issues that arise. In practice in most of the global science projects, such debate rarely occurs. When kids pose questions of students at other locations, those questions often go unanswered (Songer 1996). Many projects don't even allow kids to discuss issues that arise with one another. Instead, each class sends data in to a central authority, and the central authority does all the work of aggregating, evaluating, and presenting the data. Their conclusions are sent back to each classroom. In many ways, the organizers are engaged in a more powerful learning experience than the students. Children are serving more as technicians than scientists.

Compared to most projects of this kind, more reflection and critical debate about issues has been achieved in the CSILE (Computer-Supported Intentional Learning Environment) project. CSILE is a networked bulletin board system which structures discussions into notes and comments on those notes. Typically, a class will jointly investigate a topic. Rather than have each student complete the same assignment, students take responsibility for different aspects of the over-arching topic. The goal is to reproduce the character of scientific inquiry in a community of scientists. The designers of CSILE write:

> Can a classroom function as a knowledge-building community, similar to the knowledge-building communities that set the pace for their fields? In an earlier era, it would have been possible to dismiss this idea as romantic. Researchers are discovering or creating new knowledge; students are learning only what is already known. By now, however, it is generally recognized that students construct their knowledge. This is as true as if they were learning from books and lectures as it is if they were acquiring knowledge through inquiry. A further implication is that creating new knowledge and learning existing knowledge are not very different as far as psychological processes are concerned. There is no patent reason that schooling can not have the dynamic character of scientific knowledge building. (Scardamalia and Bereiter 1994)

CSILE is most commonly used on a local-area network, but it can be used across the Internet. CSILE or a tool like it might help organizers of global science projects to foster more reflection and critical debate among participants.

### 2.4.3 Professional Communities

While CSILE strives to give children in the classroom an activity like those of a community of adult researchers, much is being done with network technology to support actual adult research communities. This has long been the activity of a myriad of professional societies like the Association for Computing Machinery (ACM). Such societies were early adopters of email and bulletin board technologies. A great deal can happen between annual conferences. Computer networks can accelerate the pace of debate of issues, and offer individuals ongoing support in their endeavors.

**PROFESSIONAL COMMUNITIES**

| | |
|---|---|
| • AstroVR | |
| For: | Astrophysicists |
| Address: | *http://astrovr.ipac.caltech.edu:8888/* |
| • ATHEMOO | |
| For: | Theatre professionals |
| Address: | *telnet://moo.hawaii.edu:9999* |
| • BioMOO | |
| For: | Biologists |
| Address: | *http://bioinfo.weizmann.ac.il/BioMOO* |
| | *telnet://bioinformatics.weizmann.ac.il:8888* |
| • MediaMOO | |
| For: | Media researchers |
| Address: | *http://www.media.mit.edu/~asb/MediaMOO/* |
| | *telnet://mediamoo.media.mit.edu:8888* |
| • Tapped In | |
| For: | Teachers |
| Address: | *http://www.tappedin.sri.com/* |
| • The Tuesday Café | |
| For: | Writing teachers |
| Address: | *http://www.cs.bsu.edu/homepages/siering/netoric.html* |
| | *telnet://mediamoo.media.mit.edu:8888* |

Table 2.3: Professional Communities Situated in MUDs

A number of communities are now supplementing face to face meetings and mailing lists with online communities in MUDs. The first two communities to use MUDs for this more “serious” purpose were AstroVR (for astrophysicists) (Van Buren, Curtis et al. 1994) and my own MediaMOO (for media researchers) (Bruckman and Resnick 1995).[3] Compared to mailing lists, a MUD facilitates more casual collaboration. I am unlikely to send email to a colleague I’ve never met saying “I hear you do work in education. I’d like to hear more about it,” but I might say exactly that if I bumped into them in a public space online. MediaMOO functions rather like an endless conference reception for a conference on media studies.

There are a growing number of such communities. ATHEMOO is a community for theatre professionals; BioMOO is for biologists (Glusman, Mercer et al. 1996). Of particular interest are online communities of teachers. A group of writing teachers meet every Tuesday night on MediaMOO in the Tuesday Café (Fanderclai 1996), a place they built for themselves. Organizers Tari Fanderclai and Greg Siering choose a topic each week, and 15 to 60 teachers generally attend. Past topics have included the portfolio approach to writing instruction, how to equip a writing lab, and “students and the underside of the net.” Most writing teachers are under-paid, over-worked, and geographically isolated. The Tuesday Café helps them to take the process of reflecting on their practice from an annual to a weekly event. Online meetings complement face to face meetings and ongoing mailing list discussions. Tari Fanderclai writes:

> As with asynchronous forums, I am connected to people who share my interests, but MUDs provide something more. For example, the combination of real-time interaction and the permanent rooms, characters, and objects contribute to a sense of being in a shared space with friends and colleagues. The custom of using one's first name or a fantasy name for one's MUD persona puts the inhabitants of a MUD on a more equal footing than generally exists in a forum where names are accompanied by titles and affiliations. The novelty and playfulness inherent in the environment blur the distinctions between work and play, encouraging a freedom that is often more productive and more enjoyable than the more formal exchange of other forums. It is perhaps something like running into your colleagues in the hallway or sitting with them in a cafe; away from the formal meeting rooms and offices and lecture halls, you're free to relax and joke and exchange half-finished theories, building freely on each other's ideas until something new is born. Like the informal settings and interactions of those real-life hallways and coffee shops, MUDs provide a sense of belonging to a

---

[3]AstroVR began development first, and was the inspiration for MediaMOO; MediaMOO actually opened to the public first, because less specialized software development was required.

> community and encourage collaboration among participants, closing geographical distances among potential colleagues and collaborators who might otherwise never even meet. (Fanderclai 1996)

The Tuesday Cafe has been meeting since June 1993. Researchers at SRI are currently developing an online community for teachers called Tapped In (Schlager and Schank 1996a). While researchers at IBM are trying to use computer networks to replace teachers or work around them (see Section 2.2), researchers at SRI are using networks to support them, helping them to become better teachers. Organizers Mark Schlager and Patricia Schank write:

> Researchers, policy-makers, and educators view teacher professional development as a critical component of educational reform. One approach that embodies this kind of experience is the specially designed professional development institute that brings educators together around a theme or set of topics to acquire new skills and knowledge in a collaborative venue. Teachers engage in meaningful discussion with peers over several days or weeks, while interacting with a rich collection of resources. However, it is difficult to (a) scale special institutes to accommodate the large education community and (b) maintain the level of discourse and support established at the institute. Back in the classroom, teachers are once again isolated from their professional community.
>
> Our goal is to build on the strengths of these successful same-time, same-place professional development models by employing multi-user virtual environment (MUVE) technology to sustain and enrich the professional discourse, while extending access to greater numbers of educators. In service of this goal, we are developing a MUVE-based Teacher Professional Development Institute (TAPPED IN). The mission of TAPPED IN is to promote and support K-12 education reform through the establishment of a community of education professionals that is not exclusionary by virtue of geography, discipline, or technology requirements. Following exemplary teacher enhancement institutes, TAPPED IN will offer both formal events (e.g., inservice workshops, presentations) and informal ongoing activities (e.g., teacher collaboratives, case study discussion groups, apprenticeships) that teachers can access during free periods or after school. TAPPED IN will also offer services such as library facilities, bulletin boards, and e-mail. Finally, TAPPED IN is a research project intended to investigate the ways in which text-based, immersive environments initiate and sustain the growth of professional communities.

Online professional communities are exemplary knowledge-building communities. One important difference between professional communities and knowledge-building communities organized as school activities for children is that the adults have their own goals. Too often, children have educational goals imposed upon them. It would be beneficial to work towards helping students to identify their own learning objectives. Knowledge-building communities for children can learn a great deal from professional communities for adults.

### 2.4.4 Real-Time Writing

One of the earliest uses of a text-based chat system as an educational tool was with deaf students. At Gallaudet University in 1985, Trent Batson and Steve Lombardo taught a class entirely on the computer. They called this experiment "English Normal-Form Instruction". For young deaf children, English "is an experience largely limited to the classroom and lacking real-life connections"(Batson 1993). There is no mutual reinforcement between the written and the spoken word, as there is for hearing children. Using a real-time chat system in the classroom, Batson and colleagues found that they could make writing come alive:

> With a computer network and software that allows for interactive writing, deaf students can use written English not simply to complete grammar exercises or to produce compositions to be evaluated, but also to spontaneously communicate ideas that are meaningful to them with a community of other writers who are interested not in evaluating, but rather in understanding what they are saying. Written English can be used to joke and play with language, to discuss literature or serious social issues, to brainstorm ideas or collaboratively produce a draft for a paper, and to critique writing in progress. In short, written English can be used in many ways that oral English is used by hearing people. (Batson 1993)

The results of this experiment were so successful that writing teachers realized it would be beneficial for hearing students as well. Chat systems are particularly useful for helping novice writers to understand the notion of audience. Writing online, it becomes quite clear that you are writing *for someone* and need to tailor your writing to that audience. Advocates of this approach to writing instruction re-appropriated the acronym ENFI to mean "Electronic Networks for Interaction" (Bruce, Peyton et al. 1993).

A large community of teachers and researchers is continuing to explore the educational use of chat systems and MUDs for writing classes. A group of researchers at The University of Texas at Austin began developing their own software, and soon spun off a company, The Daedalus Group, to continue its development. Their product, Daedalus Interchange, is in use in a large number of schools. The computers and writing community enthusiastically uses the Internet both as an educational environment for their students, and to help themselves reflect on their practice as teachers.

No one would ever think to teach writing by lecturing to students—writing teachers have students write. While in other fields educators are struggling to increase the emphasis on learning by doing and learning through design, in writing instruction these principles have long been absolutely accepted. However, that does not mean that all pedagogical questions are answered. If learning should be self-motivated and self-directed, what do you do with students who don't want to learn? Does feedback from peers help students to become better writers, or do egos just get in the way? What power relationships exist in the classroom and how do those affect the learning process? Should we encourage students to find their own expressive style (because that is more personally meaningfully), or to conform to society's standards (because that is more economically empowering in the realities of the job market)? Who decides what constitutes "good writing"? The computers and writing community is ahead of most others communities of teachers and researchers in their exploration of many critical questions.

## 2.5 Technological Samba Schools

The approaches discussed so far have been focused to varying degrees on information—delivering it, retrieving it, and sharing it. This chapter is organized roughly in order of decreasing emphasis on information and increasing emphasis on community and the social context of learning. The last category of projects I call "technological samba schools" (see Chapter 1), a term introduced by Seymour Papert in his 1980 book *Mindstorms* (Papert 1980). In Papert's vision of a technological samba school, learning is:

- self-motivated,
- richly connected to popular culture,
- focused on personally meaningful projects,
- community based,
- an activity for people of all ages to engage in together,
- life long—experts as well as novices see themselves as learners, and
- situated in a supportive community.

Projects like The Computer Clubhouse at The Computer Museum in Boston seek to create samba-school-like communities. Kids can drop by The Computer Clubhouse after school to work with a variety of educational technologies. Mitchel Resnick and Natalie Rusk contrast The Computer Clubhouse to other projects providing community access to computers:

> "The Computer Clubhouse (organized by The Computer Museum in collaboration with the MIT Media Laboratory) grows out of this tradition, but with important differences. At many other centers, the main goal is to teach youth basic computer techniques (such as keyboard and mouse skills) and basic computer applications (such as

> word processing). The Clubhouse views the computer with a different mindset. The point is not to provide a few classes to teach a few skills; the goal is for participants to learn to express themselves fluently with new technology, becoming motivated and confident learners in the process. At the Clubhouse, young people become designers and creators—not just consumers—of computer-based products. Participants use leading-edge software to create their own artwork, animations, simulations, multimedia presentations, virtual worlds, musical creations, Web sites, and robotic constructions." (Resnick and Rusk 1996)

Real samba schools and The Computer Clubhouse are physical places. People gather there both to work on their projects and to socialize with one another. The architectural space serves as a community center for the members, providing a context for both organized activity and more casual interaction.

However, not all children live near a place like The Computer Clubhouse. Even for those who live near by, not all parents are willing to take the time to bring their children there. Only a few institutions (typically community centers and housing projects) have the resources to bring kids to The Clubhouse after school on buses. As a result, most of the members are high-school-age children who can get there via public transportation. Logistical issues have unfortunately made the clubhouse less accessible to younger children.[4]

The development of the technology of "virtual spaces" has the potential to make the idea of a technological samba school more feasible. While virtual interaction can never replace face to face interaction, network technology can be used to create communities in which people have meaningful inter-relationships, and many of the benefits of samba schools become possible. Like physical spaces, virtual spaces can provide a context for interaction among groups of people. While children don't need to travel to get to a virtual space, they do need access to a computer with a net connection. One factor limiting participation is unfortunately replaced by another.

It's worth noting that physical and virtual clubhouses are not mutually exclusive approaches. Many kids at The Computer Clubhouse participate in MOOSE Crossing. This gives them an opportunity to interact with other children not from their immediate geographic area. Interaction among members in the room is complementary to interaction with children at remote locations (see Chapter 5).

---

[4]This was the main reason that I chose to run a MOOSE Crossing after-school program at The Media Lab rather than at The Computer Clubhouse—I wanted to work with elementary and middle-school-aged children, rather than high-school students.

Calling a networked communications technology a "place" is a metaphor that helps to give participants shared expectations for how to interact with that technology, and with one another, mediated by that technology. When most people approach a computer running a piece of software, their expectations are shaped by the genre of software they are about to use. Is it a drill and practice program? Is it a spreadsheet? Is it a game? Calling a software system a "place" gives users a radically different set of expectations. Places have strong cultural associations. People are familiar with a wide variety of types of places, and have a sense of what to do there. Instead of asking "What do I do with this software?", people ask themselves, "What do I do in this place?" The second question has a very different set of answers than the first. Metaphorically calling an electronic communications medium a place lets people use their knowledge of places to help understand that communications medium. A spatial metaphor helps to create a context for the mix of playing, socializing, and learning desirable in a technological samba school.

MUDs are particularly well suited to creating technological samba schools because of their spatial metaphor, and the ways they can facilitate expressive use of words and programs. The virtual world itself is created by the members. The activity of the community becomes creating the community itself.

However, not all MUDs share qualities with samba schools. Most are violent adventure games that share few of these qualities. Even "educational" MUDs usually don't fit into this paradigm. There has been an explosion in the number of educational MUDs[5], and they represent a wide variety of pedagogical traditions. Many educational MUDs have virtual classrooms with virtual desks and virtual whiteboards where students politely raise their virtual hands to ask questions during virtual lectures. This approach is closest to distance education (discussed in Section 2.2). Other educational MUDs are experimenting with creating virtual science simulations. Such simulations could be used in a variety of ways which match with different pedagogical traditions; however, the development of this technology is in such an early state that it's not clear if any pedagogical goals are being met at all. More research is needed to evaluate its potential.

Unfortunately, MUDs are currently being used in some projects that would be better off without them. An old sophomoric joke is to take the fortune from a fortune cookie and add the words "in bed" after it. Some researchers today seem to be taking their research proposals and adding the words "in a MUD"

[5]There are many lists of educational MUDs on the net, but none of them is comprehensive or entirely up to date. The list I have found most useful is maintained by Daniel K. Schneider <Daniel.Schneider@tecfa.unige.ch> at:
*http://tecfa.unige.ch/edu-comp/WWW-VL/eduVR-page.html*

after them. This uncritical enthusiasm is unfortunate. MUDs and other forms of virtual reality technology have educational potential when used in the context of a solid pedagogical approach, and when used to take advantage of the affordances of the particular technology being used. For example, current MUDs afford the expressive use of words and computer programs. This makes them well suited to language-oriented applications such as deaf education, writing instruction, foreign language classes, and English as a Second Language (ESL) (Bruce, Peyton et al. 1993). They are also well suited to educational projects specifically about programming and other aspects of computer science. There is not yet suitable support for science simulations in MUDs in either the technology or the pedagogy—the technological infrastructure needed is not yet developed, and there are many unanswered pedagogical questions about the value of learning through simulation versus through "real" experimentation.[6] Technology can be a catalyst for meeting educational goals if the goals are put first in the design process, and technology is used appropriately to help meet those goals. All too often, the design process proceeds in the other direction, starting with what the technology can do and searching for an application.

**TECHNOLOGICAL SAMBA SCHOOLS**

- The Computer Clubhouse[7]
  *http://www.tcm.org/resources/*

- MicroMUSE
  *http://www.musenet.org/*

- MOOSE Crossing
  *http://www.media.mit.edu/~asb/moose-crossing/*

- Pueblo (formerly MariMUSE)
  *telnet://pueblo.pc.maricopa.edu:7777*

Table 2.4: "Technological Samba Schools"

Two MUD projects that stand out as samba-school-like are MicroMUSE and Pueblo (formerly MariMUSE). In both of these communities (as in MOOSE Crossing), children are encouraged to learn in a constructionist fashion—through working on self-selected, personally meaningful projects. This

[6]This question is currently being addressed by Mitchel Resnick, Robert Berg, Michael Eisenberg, and Sherry Turkle in their NSF project "Beyond Black Boxes: Bringing Transparency and Aesthetics Back to Scientific Instruments" (Resnick, Berg et al. 1997).
[7]Unlike other projects highlighted in this chapter, the Computer Clubhouse is not an Internet-based activity.

generally consists of extending the virtual world by making new places and objects.

MicroMUSE, the oldest and largest MUD for kids, has been open since 1990 and as of January 1997 had 800 members, of whom approximately 50% were children (Kort 1997). MicroMUSE was originally founded by then college student Stan Lim (Brown 1992; Kort 1997). Researcher Barry Kort stumbled on the community early in its development, and helped to shape its educational mission. MicroMUSE is also called “Cyberion City,” and is modeled as a city of the future. Navigation around the world is done in radial coordinates. The virtual world contains a number of science simulations, and scientific themes are emphasized.

MariMUSE opened originally as a summer camp activity for children organized by Phoenix College researchers Billie Hughes and Jim Walters. They chose to work with students from Longview Elementary, a school whose population is 34% Hispanic and 21% Native American. A significant portion of Longview students have limited English proficiency. Over the summer of 1993, Hughes and Walters brought children from Longview to Phoenix College for two summer sessions, each three-weeks long. Children used MariMUSE for three hours each day. The MUSE activity was particularly successful with the “at risk” students participating, several of whom appeared to develop a greater confidence in their abilities and interest in learning that carried over into the following school year (Hughes and Walters 1995; Hughes 1996; Hughes and Walters 1997).

Results from this initial summer program were sufficiently encouraging that Hughes and Walters arranged for net access to be installed at the Longview school, and the students continued to participate over the school year. The camp program was repeated the following summer, and the activity was increasingly integrated with the curriculum over the next school year (Hughes and Walters 1997). Around January 1995, Phoenix College received an ARPA grant jointly with researchers at Xerox PARC, including Danny Bobrow, Vicki O’Day, and Vijay Saraswat. The virtual world was moved from the MUSE software to MOO, and renamed Pueblo. As of January 1997, 1700 people total had participated in Pueblo at one time or another. This number includes occasional visitors, and people who have ceased their participation. Roughly 475 children were active members as of that date (O’Day 1997). Throughout its existence, Pueblo’s designers have continued to cultivate an open, student-centered learning environment (O’Day, Bobrow et al. 1996).

Both MicroMUSE and MariMUSE/Pueblo see learning through science simulation as part of their mission, as well as learning through writing and programming the virtual world. For the reasons described above, I find the latter approach more promising. Both the MUSE and MOO software are unfortunately difficult to use, and this has limited what children have been able to accomplish technically. Language design issues are discussed in detail in Chapter 3. MOOSE Crossing differs from these projects in the new technology developed for the children, and in the explicit application of the samba-school metaphor to guide its design process. The rest of this thesis discusses MOOSE Crossing as a learning environment in detail.

# 3. Designing Enabling Technologies: The MOOSE Language and the MacMOOSE Client

## 3.1 The Need for a New Language

In March of 1994, I had the opportunity to visit the MariMUSE project in Phoenix, Arizona. The previous summer, Billie Hughes and Jim Walters of Phoenix College had led a summer program where children from Longview Elementary School used a MUSE intensively, three hours a day for two three-week sessions. Seven months later, the children were still excited about their projects, and regretted not having as much time to use the MUSE as they had in the summer. They had done some impressive creative writing and building of rooms. One fifth-grade girl built a palatial mansion with flowers in each room. In real life, she lives in a homeless shelter. A Native-American boy whom teachers had considered at risk of dropping out of school built an airplane hangar. To get more realistic detail, he had read every book about planes in his school library, and asked the librarian to order more (Walters and Hughes 1994).

The children had clearly had powerful learning experiences. But something was missing. The planes didn't fly. The flowers couldn't be sniffed. None of the objects had any behaviors. None of the children I met had done any programming. One girl had made a horse with a "ride" program, but it didn't work. I sat down to look at it with her–why don't we try to fix it, I asked? She replied that an adult had written the program for her, and she had no idea how it was supposed to work.

It's not a surprise that the children were having difficulties. The MUSE language is awkward at best. Suppose that you have created an object called Rover and you would like it to wag its tail when you pet it. In MUSE, you would write:

```
@va Rover=$PET ROVER:@pemit You pet Rover.; @emit Rover wags his
   tail.
```

The name of the object can not be abstracted, but is treated as a fixed string. If you would like to be able to type "pet dog" in addition to "pet Rover," you would need to add a second command:

```
@vb Rover=$PET DOG:@pemit You pet Rover.; @emit Rover wags his tail.
```

In MUSE, each object has twenty-six lettered registers, va through vz, and a set of special-purpose, functional registers. Not only must all programs be stored on these registers, but also all data.

Now suppose you decide you want Rover to be female instead of male. To change the "his" to "her" you could retype the two lines above, or use the @edit command:

```
@edit Rover/va=his,her
@edit Rover/vb=his,her
```

The syntax is obscure, and these substitutions are prone to error. In a more complex text, it’s easy to pick too short of a target string to substitute and end up changing the wrong word (for example, changing “this” to “ther” when you meant to change “his” to “her” later on in the line.) The interface to working with the language is as much a barrier as the language itself.

Some of the children participating in MicroMUSE[1] and MariMUSE have indeed written programs. Here’s a sample program written by an eleven-year-old girl on MariMUSE using the character name “Ginji” (Walters and Hughes 1994):

```
tardis You see a unusual looking stalagmite. Type 'enter tardis to be
   able to use it! Owner: Ginji Credits: 1 Type: Thing Flags: haven
   enter_ok visible quiet
Attribute definitions: scan return #13711.return:@fo #13711=@tel
   #13035;@remit here=You see the Tardis disappear!
Va:$tower:@fo #13711={@tel #1722;@wait 15=@tr me/return}
Idesc:You are inside of the tardis.
Pennies:1 #13711.scan:$-scan:@pemit [v(n)]=> Scanning Location
   [loc(me)];@pemit [v(n)]=>REPORT:;@pemit [v(n)]=[string(-
   ,77)];look;@pemit [v(n)]=[string(-,77)]
Listen:*
Vb:$stable:@fo #13711={@tel #12375;@wait 15=@tr me/return}
Vc:$recycle:@fo #13711={@tel#13073;wait 15=@tr me/return}
Lock:Ginji(#11059Pvn)
Contents: Avalon(#12Pevc)
Home:Gingi's Mystery Cave(#13035RHvJ)
Location: Gingi's Mystery Cave(#13035RHvJ)
```

Given this cryptic and obscure syntax, it’s not a surprise that very few of the participants wrote programs. Good software can serve as a scaffolding to support children’s learning experiences (Pea 1993; Guzdial 1994). Poorly-designed software acts more like a road block.

In early 1995, Phoenix College and Xerox PARC jointly received a grant from the Department of Defense Advanced Research Projects Agency (DARPA). The MariMUSE project was renamed Pueblo, and the software was changed from MUSE to MOO (Curtis 1993). MOO is significantly more user-friendly than MUSE, and by far the best of the many currently available MUD languages—users who are not trained programmers have been able to

[1]MicroMUSE is the oldest and largest MUD for kids. It was founded by Stan Lim in 1990, and leadership of the project was soon taken over by Barry Kort (Brown 1992). Its official charter states that “MicroMUSE is chartered as an educational Multi-User Simulation Environment (MUSE) and Virtual Community with preference toward educational content of a scientific and cultural nature” (ftp://ftp.musenet.org/micromuse/Mission.Statement).

achieve more in MOO than in any other MUD language. A script to pet Rover in MOO looks like this:

```
@verb Rover:pet this none none
@program Rover:pet
player:tell(“You pet Rover.”);
this.location:announce_all(“Rover wags his tail.”);
.
```

In MUSE, to be able to “pet dog” as well as “pet Rover,” we needed to duplicate the program. In MOO, we can simply add “dog” as an alias for Rover. The word “this” in the verb declaration is an abstraction that the parser will match to any valid name for the object.

MOO, unlike MUSE, is a full programming language in which it’s possible to undertake large, complex projects. Its object-oriented nature means children can quickly create something satisfying by making a new object that inherits from an existing parent object, and then customize it and add new functionality. You can create a dog by making something that inherits from generic dog, and then program it to do new tricks. Billie Hughes highlights this as the main benefit she has observed in moving from MUSE to MOO:

> Our children do not do lots and lots of programming though many do create simple verbs. We actually found teaching verbs to be much easier than we anticipated. One 6th grader took a "generic pet" and modified the code to create a generic horse. She then set this fertile and others could adopt horses.
>
> What we have found particularly exciting about Pueblo is the inheritance/parenting/child features of an object oriented language. Hobbes [Kim Bobrow, a researcher working on the project] was able to program cats, dogs, and cars that were especially popular. Kids also loved food and clothing items. MOO let them easily create these type of objects and change the messages on the object to personalize them (Hughes 1996).

While MOO is a significant improvement over MUSE and other MUD languages, it is still too complicated for most children. Its syntax resembles a cross between C and Pascal, and any deviation from that rigidly prescribed syntax triggers an often cryptic error message. Difficult concepts are required to write even the simplest programs. Something more user-friendly is needed if children are to master it.

The MOOSE language was designed to be forgiving. For example, it’s possible to forget the quotes in programs. This only causes problems if the unquoted strings contain words used as operators in MOOSE (like “and” and “or”). In that case, MOOSE is able to detect the problem and warn the user. Table 3.1 presents the same program in MUSE, MOO, and MOOSE.

```
MUSE
@va Rover=$PET ROVER:@pemit You pet Rover.; @emit Rover wags his
   tail.
@vb Rover=$PET DOG:@pemit You pet Rover.; @emit Rover wags his tail.


MOO
@verb Rover:pet this none none
@program Rover:pet
player:tell("You pet Rover.");
this.location:announce_all("Rover wags his tail.");
.


MOOSE
on[2] pet this
  tell player "You pet Rover."
  emote "wags his tail."
end
```

Table 3.1: Petting Rover in Three Languages

Jack (boy, age 12) is the only child to date to have significant MOO experience before joining MOOSE Crossing. A transcript of a conversation in which I asked him to compare the two appears in Table 3.2. (Jack was aware that he was talking to the designer of MOOSE Crossing, and his comments must be seen in that light.)

## 3.2 The Design of the MOOSE Language

Work on the MOOSE Crossing project began in September of 1992. It was referred to as “The MUD for kids I’m working on,” until I came up with name MOOSE in June 1993. The client implementation was begun in 1992, and the server/language in November 1993. Children first used MOOSE in October 1995, as part of the Media Lab’s 10th birthday celebration.

The core of the MOOSE language was designed in a series of weekly meetings between myself and my advisor, Professor Mitchel Resnick. MIT undergraduates Albert Lin, Trevor Stricker, and Austina Vainius helped design many of the finer details in innumerable impromptu discussions at my white board. Pavel Curtis, Randy Farmer, and Brian Silverman

[2]With the Epistemology and Learning Group’s strong roots in the Logo community, it was tempting to start programs “to tickle” rather than “on tickle”. However, in general, in MUDs an object holds scripts for things that can be done to it. A bear’s tickle script enables people to tickle the bear, not the bear to do the tickling! This is better conveyed by “on” rather than Logo’s traditional “to”.

contributed design ideas on visits to the lab, and via email. Jon Callas and Dean Tribble also contributed ideas via email.

```
Amy says 'so I wanted to ask you... now that you've been here a
   while, what do you think of coding in MOOSE versus MOO?'
Jack says, 'moose is sooooo much simpler!'
Jack says, 'easier and a lot more fun'
Amy says 'in what ways is it simpler?'
Jack says, 'well, instead of this.location.announce_all it is just
   'announce''
Amy nods
Jack says, 'and i can make my own scripts, not just copy lines of a
   help file and changing them'
Amy says, 'cool!'
Amy says, 'did you notice the difference in how property references
   work?'
Jack says, 'yes, I also like how the edit script buttons and help
   buttons are seperate, easy windows'
Amy nods and listens
Amy says, 'anything you don't like?'
Jack says, 'I don't like when i crash every time i get red letters in
   programing a script.'
Amy laughs.  "Yeah, that was awful!  But you know we fixed it,
   right?"
Amy says, 'the new version of MacMOOSE doesn't do that'
Jack says, 'wow! I will have to convert!'
Amy says, 'definitely!'
Amy says, 'anything else you don't like?'
Amy says, 'or that you would change?'
Jack says, 'say where do i find the new version? hmm... somthig else
   i dont like...'
Jack says, 'i can't think of anything else!'
Amy says, 'the new version is on my web page where you got the first
   version'
Amy smiles
Amy says, 'http://asb.www.media.mit.edu/people/asb/'
Amy says, 'Does anything other than the software seem different to
   you?'
Jack says, 'hmm, i have been to many many muds, and moos, the only
   one that is close to comparing is mny old moo, du. But this is
   diffrent then that because of the @ commands...'
Jack says, 'and the fact that is is for kids'
Jack says, 'is is = it is'
Amy nods
Amy says, ‘do the people seem any different?’
Jack says, 'more friendly, i seem to interact with them better...'
```

Table 3.2: Jack’s Opinion of MOOSE versus MOO

MOOSE is built on top of MOO, and many aspects of its design are reacting to MOO, modifying it based on experience with the design of Logo and StarLogo in particular. Several principles emerged in the process of designing the language (See Table 3.3). I’ll discuss each in turn.

**Design Principles:**
- Have a gently-sloping learning curve.
- Prefer intuitive simplicity over formal elegance.
- Be forgiving.
- Leverage natural-language knowledge.
- Avoid non-alphanumeric characters wherever possible.
- Make essential information visible and easily changeable.
- It’s OK to have limited functionality.
- Hide nasty things under the bed.

Table 3.3: Design Principles for the MOOSE Language

Parts of this chapter critique the design of MOO in detail. It should be noted that MOO is a production-quality, full programming language. MOOSE is quirky and incomplete. It continues to be extended and modified based on feedback from children. MOOSE is also roughly an order of magnitude slower than MOO. My goal in this critique is to bring forward design issues of broader interest—not to be critical of MOO.

### 3.2.1 A Gently-Sloping Learning Curve

For a programming language to be usable by young children, simple things need to be simple. In the earliest design meetings about the MOOSE language, we focused on how to make the learning curve as gentle as possible. Our first design decision, and one from which many other design decisions followed, was to make the programming language and the command line language as similar as possible. That way, kids can try a command out and put it in their program if it works. In MOO, the command line and programming languages are quite different. For example, in both MOO and MOOSE, to take an action (like smile or wink), you use “emote”. If I type “emote laughs” everyone in the room sees “Amy laughs”. Emote is typically one of the first commands that people on MUDs learn. In MOOSE, you can use that same command-line command in your program. Objects as well as people can emote. For example, Chester does this on his “computer” object. Chester is seven years old, and one of the youngest children to use MOOSE Crossing to date. When you type “eat computer” it tells everyone in the room “computer gobbels up a disk happoly.” Here is the program:

```
on eat this
  emote " gobbels up a disk happoly."
end
```

In MOO, this same program would be:

```
   this.location:announce_all(this.name + “ gobbels up a disk
      happoly.”);
```

In MOOSE, children can learn commands to use in conversation and then use those same commands in their programs. In MOO, an entirely new set of commands with different syntax needs to be learned for even simple programs.

Making the programming and command line languages the same helps more advanced MOOSE programmers as well as novices. For example, a child learning to use lists can try out list manipulation commands at the command line first seeing exactly how they work before trying to make them function in the context of a longer program.

The benefits of giving programmers immediate feedback can be traced back to the first interactive programming language, JOSS (Baker 1981). JOSS is cited as an influence by the developers of most early languages targeted at non-experts, including Logo (Feurzeig 1996), Smalltalk (Kay 1996), and BASIC (Kurtz 1981). There is no concept of a “command line language” distinct from a “programming language” in most interactive languages. The distinction was added in some MUD languages like MOO and LPC (the C-like language in which LPMUDs are constructed) to make a distinction between metaphorically acting in the virtual world and programming it.[3] This allows taking actions to be more natural-language-like, while programming looks more like typical programming languages. However, it adds a significant hurdle for people trying to learn to program, essentially removing the benefits of having an interactive language.

The similarity between the command line and the programming language is just one example of the many ways we tried to make the slope of the learning curve gentle. This principle was central throughout the process of language design.

### 3.2.2 Intuitive Simplicity Versus Formal Elegance

The second and perhaps most important design decision was to favor intuitive simplicity over formal elegance. In “Boxer: A Reconstructable Computational Medium,” Andy diSessa comments that to create a popular medium, “a computer scientist’s or a mathematician’s measures of simplicity are simply not an issue. A better criterion is accessibility to a seven-year-old child” (diSessa and Abelson 1986).

[3]In the MUSE language, the command line and programming languages are identical. However, the integration of support for their distinct requirements is awkward. For example, to give something to someone in MUSE, you type “give person=object.” In MOO or MOOSE, you “give object to person.” More sophisticated parsing in MOOSE makes a smoother integration of command-line and programming language requirements possible. However, there is a trade-off: MOOSE’s pattern-matching algorithm makes it substantially slower than MUSE.

There are compelling arguments in favor of paying attention to formal elegance. The idea is that through mastering some perhaps initially problematic construct, students will gain access to a way of thinking that will deepen their understanding in the long run. However, the problem arises: what happens if children never master the construct? Furthermore, does that transfer to broader concepts really occur, and are the broader concepts people are striving for really the right ones to be concerned about? What is this “formal elegance”? I agree with diSessa that the notion of elegance is a byproduct of an adult aesthetic, and largely irrelevant to children’s needs.

The design of a language is primarily about tradeoffs. Given a choice between elegance and intuitiveness, we resolved to lean as far as possible towards intuitiveness. If you can have both, that’s preferable; however, sometimes it’s necessary to chose. Consider the following difference between Logo and StarLogo. Table 3.4 shows how you define a variable foo, set its value to 3, and retrieve the value of that variable in Logo, StarLogo, and MOOSE.

| **Language** | **Set Variable** | **Access Value** |
|---|---|---|
| Logo | `make "foo 3` | `:foo` |
| StarLogo | `set foo 3` | `foo` |
| MOOSE | `set foo to 3` | `foo` |

Table 3.4: Variables in Three Languages

In Logo, the quote before the variable definition makes it clear that foo is a symbol, to which we are assigning a value. The colon indicates that we are accessing the value of the variable and not merely referring to the symbol. The semantics are very clear, and highlight some underlying computer-science concepts. However, in years of working with children with Logo, Mitchel Resnick has found that the quote and colon are some of the most common causes of difficulty and error for children. It’s not clear whether the broader concepts about symbols and quoting are commonly learned, but it is clear that these syntactic requirements slow down children’s progress on their Logo projects (Resnick 1993).

In the StarLogo implementation , these concepts are a bit blurred. The technical explanation is that “set” is a special form that doesn’t require quoting of its first argument. On declaring a new variable, an accessor function by the same name is automatically created which returns the value of that variable. This technical explanation is almost certainly lost on children using the language—they don’t think about things like what a symbol is because there is no need to. There is less special syntax to learn and fewer things that can easily be gotten wrong (Resnick 1997).

The MOOSE version is almost identical to the StarLogo version. The added preposition “to” is consistent with MOOSE’s natural-language-like style. This has the advantage that it draws on children’s existing natural-language knowledge. It has the disadvantage that it may mislead them that more English-like commands work than actually do.

Which of these styles is preferable? In theory, you can make a case for any of them. However, working with children in practice, it seems clear that the Logo style is idealistically naive. The StarLogo and MOOSE approaches let children accomplish more. The children are better able to express their ideas, and therefore create more complex final products.

A cynic might ask, “If we’re going to make things as simple as possible for the children, how about giving them an expert system that would create the entire project for them? Isn’t the real educational value in doing the hard work of understanding some not immediately intuitive concepts?” To this I would reply, the educational benefit is not in the sophistication of the product; it is in the ability to express and refine a complex idea. If the child gets stalled early on by syntactic difficulties, the entire learning experience can be stalled. If the child is able to express ideas fluently, the child’s learning experience will progress as the complexity of his or her project progresses.

These same issues emerged in the design of Microworlds Logo. In 1991, Logo Computer Systems International designed a new version of the Logo programming language called Microworlds Logo, and members of the Media Lab’s Epistemology and Learning Group met several times to critique its evolving design. A heated debate ensued at MIT and in the broader community about the inclusion of paint tools. When most people think of children programming in Logo, they think of children learning geometry through drawing simple shapes. To make a square of size 100, you’d need to type: “repeat 4 [fd 100 rt 90]”. The child’s desire to make a picture provides a context that gives certain geometrical concepts new relevance. In Microworlds Logo, the child can simply select a paint tool from the menubar to draw a square or rectangle. Purists argued that this was robbing Logo of powerful potential for learning. If Logo were to be transformed into a simple paint program that would indeed be the case. Paint programs generally afford artistic learning experiences but not mathematical ones.[4] However, Microworlds is not merely a paint program—it is still the Logo programming language, and compared to other versions of Logo adds a series of features (buttons, fields, multiple turtles, multiple changing turtle shapes, etc.) that afford whole new categories of projects: simple animation and video games.

[4]Noteable exceptions are Mike Eisenberg’s SchemePaint system (Eisenberg 1995) and David Shaffer’s work with Escher’s World (Shaffer 1996).

Paula Hooper worked with children at Paige Academy in Roxbury, Massachusetts for six years. For the first two years, her students used LogoWriter, a traditional version of Logo that works on Apple IIgs computers. The school then got Macintosh computers. For just a few weeks, the computers weren't equipped with Logo, but did have copies of Kidpix, a drawing program made by Broderbund Software. The students became very involved with making pictures in Kidpix. Soon after, the school received copies of Microworlds Logo. At first, Hooper had difficulty getting the children to stop drawing pictures in Kidpix and start working with Microworlds. However, she reports that once they realized what they could accomplish with Microworlds Logo, the children chose it over the paint program. Hooper worked with the same children using Microworlds for the following four years. The Paige Academy students are allowed to work on projects of their own choosing. Most use Microworlds to tell animated stories, and make their own video games. They have become enthusiastically involved in making much more complex projects than they were ever able to make in LogoWriter. This is of course partly because they are now older; however, Hooper also believes the affordances of the tool made much more exciting projects possible for them. They are learning less about geometry, but more about computational ideas (like procedural abstraction), and most importantly more about the process of designing and following through on a large, complex project. In LogoWriter, Hooper notes that even the simplest project was a substantial undertaking. The formalistic aesthetic that children should be forced to make squares out of combinations of ninety-degree turns is misguided. While some simple things (like making circles and squares) are done automatically for you in Microworlds, this makes more complex projects possible. The richest learning occurs through the process of designing a complex project that the student finds personally compelling (Hooper 1997a; Hooper 1997b).

An example of a trade off between formal elegance and intuitive simplicity in MOOSE is the question of quoting strings. In a MUD, it's natural not to quote most commands at the command line. You want to type:

```
say Hi there!  How is your project coming?
```

Not:

```
say “Hi there!  How is your project coming?”
```

However, we wanted to permit a more complicated syntax at the command line than is normally allowed in MUDs:

```
say “Today’s secret word is ” + my secret_word
```

You wouldn't normally use property references like that in conversation, but you might want to use them in programs. Therefore, you should be able to try them out at the command line.

Prohibiting unquoted strings at the command line would enforce a neater style, and help children to understand the notion of quoting. If we made that decision, the concepts would be clear: words always go in quotes; commands and variables don't. That would be the most elegant solution. But frankly, it's a hassle—no one really has spontaneous text-based conversations that way. And do you really want to start giving kids error messages every time they forget a quote?

The opposite extreme, never requiring quotes, causes implementation problems. In particular, we chose to make certain English words behave as operators in MOOSE, for example: *is*, *are*, *and*, *or*. How could you tell whether someone wanted to test equivalence or just use the word "is"?

We arrived at a complex compromise. At the command line, commands can be in one of two styles:

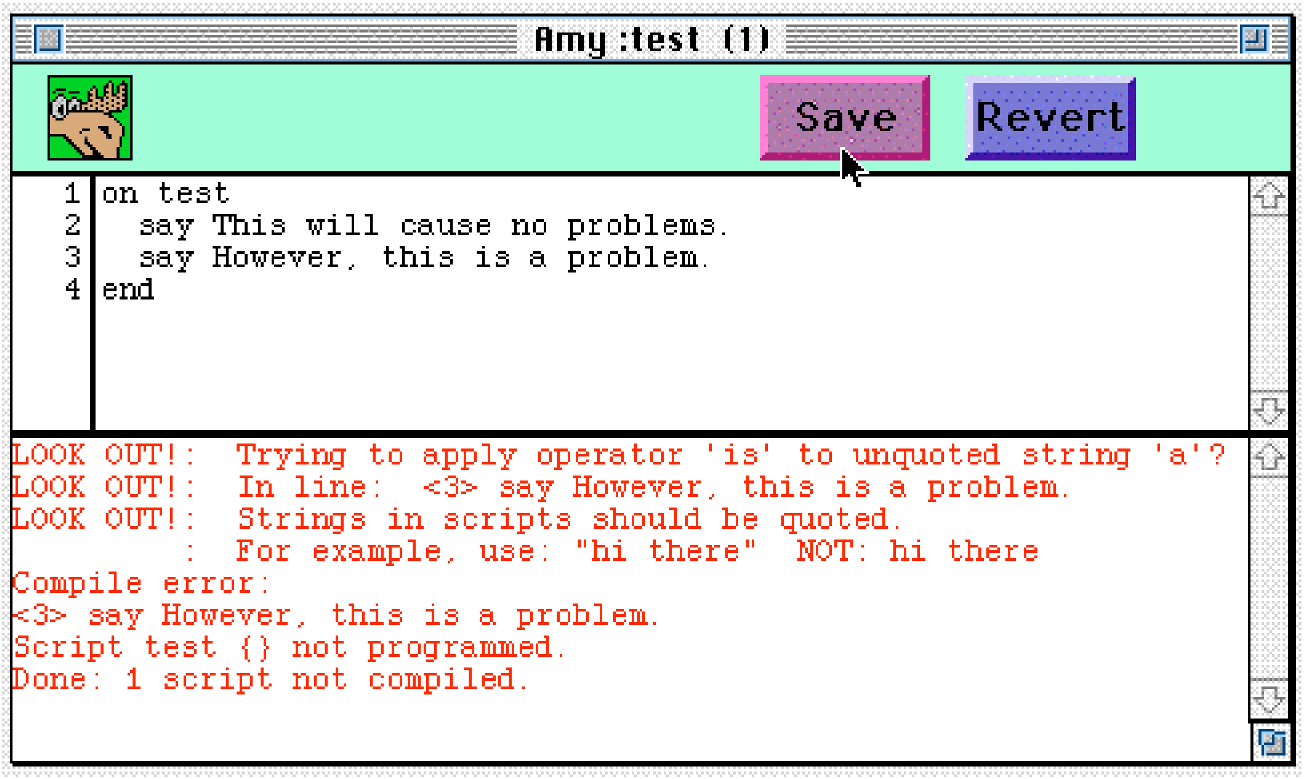
```
say In style one, the entire string is unquoted.  Since the strings
   around the operators are not quoted, the parser concludes that
   this must be an unquoted string, and leaves it alone.

say "In style two, the strings are quoted, and " + my adverb + "
   combined together."
```

In the first example at the command line, if you forget a plus or a quote, it is no longer a well-formed compound expression, and the system will treat the entire expression as an unquoted string.

In programs, you can get away with having unquoted strings, if they contain no infix operators. If they do, you will get a warning message when you compile them. An example appears in Figure 3.1. Feedback warning you about the problem appears in red in the feedback pane of the script editor window, immediately under your code. The system's ability to warn the user of potential problems helps avoid the problem of misunderstandings common to "do what I mean" (DWIM) systems.

The details of this solution may sound complicated, but to the user it amounts quite simply to: It's better to use quotes, but you can often get away without them.

In use to date with children, on the whole it is the younger children (nine years old and younger) who tend not to use quotes in programs. The decision to allow sloppy quoting has made MOOSE accessible to a wider range of ages and ability levels. The fact that the older children (ages ten to thirteen) tend to quote their strings supports the fact that this sloppiness has led to no fundamental conceptual confusion.



```
on test
  say This will cause no problems.
  say However, this is a problem.
end
```

```
LOOK OUT!:  Trying to apply operator 'is' to unquoted string 'a'?
LOOK OUT!:  In line:  <3> say However, this is a problem.
LOOK OUT!:  Strings in scripts should be quoted.
         :  For example, use: "hi there"  NOT: hi there
Compile error:
<3> say However, this is a problem.
Script test {} not programmed.
Done: 1 script not compiled.
```

Figure 3.1: Unquoted Strings in a MOOSE Program

After reading a draft of this chapter, Danny Bobrow suggested that it would be desirable to have the system volunteer to correct the program. Instead of simply allowing quoted strings, the client could prompt the user asking whether they should be quoted. This might have significant pedagogical advantages. Making this approach successful would require the suggestions to have a high level of accuracy, and never to be repeated if rejected. Otherwise, the system's active interventions could be annoying. Bobrow's proposal merits further exploration in future systems.

### 3.2.3 Be Forgiving

MOOSE's tolerance of unquoted strings illustrates another important design principle: be forgiving. It's very often possible to anticipate certain common user errors and adopt a "do what they meant, not what they said" attitude. Being forgiving about quoting has made MOOSE accessible to younger children than would otherwise have been possible. In work with kids using MOOSE to date, we've found that they often begin by not quoting strings, but learn to do so as time goes on. Allowing unquoted strings in programs helps them complete their initial projects more easily. Success in those initial projects helps to deepen their long-term involvement.

The disadvantage of this approach is that it sometimes makes it difficult to give the programmer helpful error messages. For example, consider this program:

```
set result to 3
say resilt
```

This MOOSE program will say the word “resilt” (note the typo) instead of saying 3, the value of the variable “result”. In a language that doesn’t allow unquoted strings, the second line could generate an error message because “resilt” is undefined. MOOSE assumes “resilt” is an unquoted string. However, MOOSE is able to give somewhat better feedback to this error:

```
set result to 3
say “The answer is ” + resilt
```

Compiling this program gives this feedback:

```
LOOK OUT!:  Trying to apply operator '+' to unquoted string 'resilt'?
LOOK OUT!:  In line:  <3> say "The answer is " + resilt
LOOK OUT!:  Strings in scripts should be quoted.
         :  For example, use: "hi there"  NOT: hi there
Script 'typo_example' programmed.
Done: 1 script compiled.
```

An unquoted string on its own is undetectable; however, applying an operator to an unquoted string generates a warning message. The error message is not ideal, but at least it alerts the user to the correct location of the problem. The “do what they meant” design philosophy allows us to give good feedback most but not all of the time.

### 3.2.4 Leveraging Natural-Language Knowledge

One potentially dangerous decision we made was to leverage off of children’s natural language knowledge. For example, in MOO, property references are of the form `<object reference>.<property name>`. MOOSE property references use an English-like possessive notation. The possessive notation is natural and children pick it up without having it explained to them. Their natural language knowledge makes the use of property references easy.

| | |
|---|---|
| MOO: | <object reference>.<property name> |
| Examples: | #99.age |
| | #99.owner.name |
| | |
| MOOSE: | <my / object’s / objects’> <property name> |
| Examples: | my name |
| | Ello’s name |
| | Ello’s owner’s description |
| | Snuggles’ description |

Table 3.5: Property References in MOO and MOOSE

The first language to use English-like syntax in order to be more accessible to non-professionals was COBOL. Jean Sammet, one of the designers of COBOL, writes:

> Although from the very beginning COBOL was concerned with “business data processing,” there was never any real definition of that phrase. It was certainly intended (and expected) that the language could be used by novice programmers and read by management. We felt the readability by management could and would be achieved because of the intended use of English, which was a fundamental conclusion from the May 1959 Pentagon meeting. Surprisingly, although we wanted the language to be easy to use, particularly for nonprofessional programmers, we did not really give much thought to ensuring that the language would be easy to learn; most of our concentration was on making it “easy to read” although we never provided any criteria or tests for readability. (Sammet 1981)

In the design of MOOSE, we put as much emphasis on “learnability” and “writeability” as in “readability.”

More than twenty-five years after the design of COBOL, the designers of Hypertalk had similar goals and strategies. When asked about the language ancestors of Hypertalk, designer Bill Atkinson replied “The first one is English. I really tried to make it English-like” (Goodman 1988). Ted Kaehler, another member of the Hypertalk design team, comments that “One principle was ‘reads as English, but does not write as English.’ Like an ordinary programming language, it depends on exactly the right syntax and terms” (Kaehler 1996).

In designing MOOSE’s natural-language-like syntax, we drew most heavily on MOO’s command-line language (Curtis 1993), but also borrowed directly from Hypertalk. For example, MOOSE borrows Hyper Talk’s use of the variable “it” to refer to the last value returned.

The risk of making a computer language like a natural language is that people will assume more natural language constructs work than really do. On the whole, this has not proved to be a problem. MOOSE commands have a consistent syntax: each command begins with a verb. The arguments a command takes are readily viewable with system commands like "examine," "which," and "show."

The slippery slope of natural language has proved to be a problem in only one area: conditionals. While the syntax of most commands roughly follows a simple "verb direct-object preposition indirect-object" pattern, the syntax of the Boolean clauses of if, elseif, and while statements is much more complicated. Most of the conditionals the kids have written so far have been simple enough to pose few problems. But consider these lines (from a script by me, the "teach" script on Generic Teachable Object):

```
if prop member_of my teach_locked and player is not my owner and not
   player's admin
  tell player "Permission denied."
  return
endif
```

This does indeed have a specific syntax: clauses are separated by the conjunctions "and" and "or"; individual clauses are usually of the form <argument> <operator> <argument>, or simply <argument>. However, this syntax is more complex than other MOOSE constructs; consequently, it's easier to start assuming arbitrary English statements will work.

The most common mistake we observed was for kids to put a "not" in the wrong place. For example, they might write "A is not B". This leads to some potential confusion as to whether the negative applies to the element B or to the clause. To clear this up, I made the compiler always assume a negative applies to the clause—they presumably meant "not (A is B)". Negating an individual element is an advanced concept that I saw no advantage in introducing. (It's still possible to do so—you just need to use parentheses: "A is (not B)".) The second most common mistake was to add an extra "is." For example, they might write "item is member of list" instead of "item member of list." The words "is" and "member" are separate operators, and their composition is nonsensical—the programmer clearly meant just to use the "member" operator. The easy fix for this is simply to eliminate any instances of "is" before another operator. This was easy to do in the compiler. Another simple fix was to allow certain operators like "member_of" be written as either one or two words. These technical improvements have greatly reduced the number of errors kids encounter. However, it still remains somewhat difficult to debug bad conditional expressions. This is an area in which the language could be improved.

Over all, the similarity of MOOSE to natural language has proved to be a good design decision. Kids often look at other kids' programs and understand them without having them explained. They are immediately readable. One of the most common learning techniques I've observed kids using is to start with a simple variation on another child's program. Later they progress to making increasingly original creations. This immediate understanding of programs is a result of the language's natural-language-like structure.

### 3.2.5 Avoid Non-Alphanumeric Characters

One of the easiest design decisions we made was to avoid non-alphanumeric characters wherever possible. Kids aren't familiar with them, and identifying them and typing them pose problems. In particular, many commands in MOO are preceded by the character @. The logic behind the @ command in MOO is this: commands to the programming environment use @s; commands that simulate taking actions in the virtual world do not. In practice this rapidly breaks down. Many commands are user-defined, and not every programmer understands or respects the convention. Furthermore, whether a command is equivalent to taking action in the virtual world is not always clear. For example, if you're reading a political ballot on LambdaMOO but the ballot object is not in the room but is defined system-wide, are you reading it or @reading it? What about if you're reading your MOO mail? The former is "read" and the latter is "@read". It's hard to keep them straight. MUSE also uses @s, but with a different rationale. In MUSE, commands that have a side effect to change something in the database are preceded by an @. This also is somewhat ambiguous. For example, walking around the virtual world does not require an @, yet that does change something in the database—your location. Teleporting on the other hand does require an @ in MUSE. The very fact that the convention for when to use an @ varies between languages also significantly contributes to the confusion. People frequently use multiple environments, and may not realize that the convention differs. We resolved this in MOOSE by eliminating @s altogether.

One common problem for novice programmers is the tendency to confuse using an equals sign for assignment and for equivalence. In MOOSE we avoided this problem by using "set" for assignment (i.e. "set my age to 30"), and English words for equivalence: is, are, isn't, and aren't. Other operators are also converted to words. MOO's operator "&&" becomes "and"; "||" becomes "or".

In addition to being unfamiliar and hard to type, non-alphanumeric characters look "high tech." Much of MOOSE is learned by looking at sample programs. Letters are familiar to children, and less threatening. Programs filled with special characters may tend to scare off new users by making coding look hard.

In a visit to a classroom of “Title I” students (students who are more than two years behind in their reading level) using MOOSE Crossing, I noticed that those students had particular difficulty typing unfamiliar characters. Characters that require the use of the shift key appeared to be particularly problematic. While I eliminated most special characters from MOOSE, a few are still used occasionally, particularly underscores. Observing the Title I students, it was clear that this was a mistake. They will be removed in the future. While they appeared to be struggling with simple reading and writing, the special characters seemed to be an even bigger hurdle.

English words have every-day-use meanings that often make it easier for them to understand and remember the computer meanings of those words. Special characters usually don’t have those ordinary meaning, so their computer meanings are more easily confusable. This became abundantly clear with one special character we did include in MOOSE, single-quote ('). The two most commonly used commands in MUDs are “say” and “emote.” They are used so often that it’s common in most MUDs to allow them to be abbreviated as double-quote (") and colon (:) respectively. In MOOSE, we wanted to avoid confusion between the abbreviation for say and quoted strings, so we made the abbreviation be single-quote ('). One common point of confusion for people new to MUDs is the distinction between saying something and doing it. Contrast the following three commands:

A.
```
--> say down
You say 'down'
```

B.
```
--> 'down
You say, 'down'
```

C.
```
--> down
You climb down the rope to the platform.
```

Saying the word "down" is not the same thing as trying to actually go down, changing your location in the virtual world. It's easy to see the different between A and C above. It's harder to see the difference between B and C. Here's an edited excerpt from a confusing afternoon when a class of twenty students connected to MOOSE Crossing for the first time. Experienced users Miranda (girl, age 10-11) , Rufus (boy, age 12), and Newton (teacher, male, age 41) came by to help the new users. Austina and I were also there. Not long after all twenty students connected for the first time, this conversation ensued:

```
Amy says, 'wow, quite a crowd!'
jj says, 'Who is Pumpernickle''
cj says, 'cj here'
```

```
Bill says, 'pumpernickel''
Miranda says, 'A dog'
Tim says, 'hi Tag'
Amy [to jj]: Pumpernickel is my dog. Try this: pet pump
Newton floats above the crowd, nervous and shy.
Rufus says, 'You guys can have dogs too.'
jj says, 'Can we pet him/her''
Miranda says, ''Yeah.'
Rufus says, 'It's up to Amy...'
Amy says, 'you can tickle her too'
Bill says, 'BeeBee''
jj says, 'pet Beebee''
Tag says, 'pet beebee'
isaac says, 'hi pet beebee''
Tim says, 'pet BeeBee''
Amy [to jj]: don't say it, do it. Don't put the ' in front
Bill says, 'pet BeeBee'''
```

BeeBee is their teacher’s virtual pet dog. Presumably, the students’ teacher is suggesting they pet BeeBee. This doesn’t succeed for two reasons. First, they are saying the words instead of taking the action by accidentally prefacing their commands with a single quote. Second, BeeBee isn’t in the same room in the virtual world as the students.

```
Rufus says, 'If anyone wants to see something cool, type 'enter
   Sparky'.'
jj says, 'pet BeeBee''
[Rufus types: enter Sparky
[From Sparky III] Duggan wags his tail at Rufus.
Bill says, 'Sparky''
isaac says, 'sparky''
Miranda pets Pumpernickel.
Pumpernickel wags her tail and licks Miranda's hand.
Jermaine says, 'Hi Lara''
Tim says, 'enter Sparkey'
Amy says, 'don't put a 'say' or a quote in front of the word pet'
Alana sniffs Rufus curiously.
cj says, 'Bill what class are you in''
jj says, 'enter Sparky''
Miranda pets Pumpernickel.
Pumpernickel wags her tail and licks Miranda's hand.
Lara says, 'pet beebee'
Rufus says, 'type 'enter Sparky''
Miranda says, 'This is too confusing!'
Bill says, 'enter Sparky''
jj says, 'enter Sparky''
isaac says, 'sparky''
Austina goes home.
Alana follows after Austina.
Amy [to Bill]: don't put the ' or say in front of it. Type it just
   like this: enter sparky
Rufus says, 'No! Type it on the keyboard!'
isaac says, 'look''
jj says, 'look''
Tag says, 'look''
Lara says, 'look''
Tim says, 'By austina''
```

```
Rufus says, 'Thank you Amy!'
Jermaine says, 'look''
[At this point, Bill gets it right and enters Sparky]
[From Sparky III] Duggan sniffs Bill curiously.
Bill You climb onto the main deck.[5]
```

A number of things contributed to making this encounter confusing. The sheer number of students was a significant factor. Their teacher had only tried MOOSE Crossing a couple times herself. In fact, she was giving the class incorrect instructions out loud, telling them to type commands both starting with and ending with single quotes. (This led to the extra single quote at the end of most things the new members said.) She told them to do this for all commands, not just things they wanted to say. Once this misconception was created, it was hard to correct, particularly since single quotes are used to delimit ordinary conversation. When you give someone instructions on what to do, single quotes wrap what you are saying. It therefore is difficult to talk about when to use single quotes. The confusion persisted for some of the students even when they returned for a second session the following week. Perhaps the most important contributor to the confusion was the design decision to allow the use of single-quote as a shortcut for the “say” command. The teacher was giving the students explicit wrong directions; however, the teacher herself might not have been confused if the command had been clearer. Furthermore, even if the teacher had given them incorrect instructions, the students would have been more likely to figure out what was wrong if they had been using the English word “say” instead of an ambiguous punctuation mark. All the students were using single quote, not “say,” in this conversation. It's easy to understand how they came to type these incorrect commands:

```
‘pet BeeBee’
‘look’
‘enter Sparky’
```

It’s harder to imagine them making the equivalent mistakes if they knew only the “say” form:

```
say pet BeeBee
say look
say enter Sparky
```

New MOOSE Crossing members usually learn to use “say” first, and only learn the single quote shortcut later. This significantly reduces the potential for confusion. This level of confusion is not typical. However, it still reinforces the broader point: non-alphanumeric characters are best avoided if possible.

[5]Several lines were edited out of this transcript for clarity and conciseness.

### 3.2.6 Make Essential Info Visible and Easily Changeable

In some programming languages, for example C and MOO, information about a function is stored separately from the code for the function itself. This separation is confusing. While trying to fix a problem, users often find that the information needed to fix the problem is not in front of them. Remembering to check the information about the function (or property) as well as its contents is a debugging strategy that most new programmers are slow to master. It's advantageous to keep essential information with the code itself.

In particular, both MOO verbs and MOOSE scripts have arguments. In MOO, these must be declared when the verb is declared:

```
--> @verb me:jump this on any
Verb added (8).

--> @verb me:jump this none this
Warning:  Verb `jump' already defined on that object.
Verb added (9).

--> @verbs me
;verbs(#75) => {"enlist", "random_player", "hack", "words", "jump",
   "jump"}
```

MOO argument specifications must be three items long (direct object, preposition, indirect object), and the preposition choices are drawn from a fixed set of alternatives (Curtis 1993). The special argument specifier "this none this" is defined as meaning that the verb is not intended to be called from the command line, but only from other verbs. Working with multiple verbs of the same name is extremely difficult, because the interface has few affordances for indicating which verb of a given name you mean. Listing the code for a verb, you do not see its arguments. To see its arguments, you must use a special command like "@show" or "@display". To change the arguments, you must use another special function: "@args"; you can't use the same mechanisms you use to change the code.

In MOOSE, argument specifications can be of any length, and any word may be used as a constant. A script's argument declaration is simply the first line of the program. It's clearly visible as you edit the program, and can be changed by the same mechanism you use to edit the program. Multiple scripts with the same name can simply be placed one after the other in the script body, with no confusion between them. For example, here's the "feed" script on Generic Penguin by Rachael (girl, age 12-14):

```
on feed this
  tell context "You feed " + my name + " fish."
  announce_all_but context context's name + " feeds " + my name + "."
  set my ishungry to 1
  emote "glups the fish down hungrily."
  set my last_feed_time to time
end

on feed this string
  tell context "You feed " + my name + " a " + string + "."
  announce_all_but context context's name + " feeds " + my name + " a
   " + string + "."
  set my ishungry to 1
  announce_all my name + " glups the " + string + " down hungrily."
  set my last_feed_time to time
end

on feed this herring
  tell context "You feed " + my name + " pickled herring."
  announce_all_but context context's name + " feeds " + my name + "
   pickled herring."
  emote "puckers up and spits the herring out."
  emote "blahs!"
end

on feed this shrimp
  if my diet is 1
    tell context "You can't feed " + my name + " shrimp today because
   " + my name + " is on a diet."
    announce_all_but context context's name + " tries to feed shrimp
   to " + my name + " but " + my name " is on a strict diet!"
  else
    set my ishungry to 1
    tell context "You feed " + my name + " a ton of shrimp!"
    announce_all_but context context's name + " feeds " + my name + "
   a load of shrimp."
    emote "gobbles the shrimp down in seconds."
    set my last_feed_time to time
  endif
end
```

```
on feed this eggs on toast
  set my ishungry to 1
  tell context "You feed " + my name + " a soft boiled egg on toast."
  announce_all_but context context's name + " feeds " + my name + " a
   soft boiled egg on buttered toast."
  emote "polietly picks up the toast with " + my name + "'s left
   flipper and nibbles the end like so."
  announce_all_but context "After chewing, " + my name + " thanks " +
   context's name + "."
  tell context my name + " says, 'Thank you ever so much in offering
   me this delightful meal. I would be ever so obliged if you would
   be so kind as to give me a cup of tea to wash it down.'"
  set my last_feed_time to time
  set my askfortea to 1
  fork 10
    if my askfortea is 1
      tell context my name + " says, 'Thanks alot!'. (You didn't give
   " + my name + " any tea!)"
      set my askfortea to 0
    endif
  endfork
end

on feed this tea
  if my askfortea is 1
    tell context "You give " + my name + " a cup of tea."
    set my askfortea to 0
    announce_all_but context context's name + " gives " + my name + "
   a cup of tea."
    emote " picks up the cup from the saucer, and takes a sip."
    tell context my name + " says, 'Thank you ever so much for the
   tea. I feel honored that you decided to honor my request for
   tea.'"
  else
    tell context "You give " + my name + " a cup of tea."
    announce_all_but context context's name + " gives " + my name + "
   a cup of tea."
    emote "slurps up the tea."
    emote "nods in thanks."
  endif
end
```

Andy diSessa argues that programming languages should be “tuned toward small tasks—the ability to implement simple ideas easily is much more important than the ability to do complex tasks efficiently”(diSessa and Abelson 1986). The ability to have lots of short scripts of the same name helps break tasks into parts more clearly.

### 3.2.7 It’s OK to have Limited Functionality

In our first serious discussion of the design of MOOSE, Pavel Curtis looked at me squarely and said: “If you can do everything in MOOSE that you can in MOO, I’ll be disappointed in you.” The designers of Logo pride themselves on the fact that Logo is a full programming language that you could use for professional software design if you wanted. The ideal is that a tool should have no initial barrier and no ceiling—no limits on what a kid can achieve. While this is an inspiring vision, the truth is that few kids actually reach those higher levels. If advanced functionality comes at no cost, it's certainly desirable. However, in the design of MOOSE when we encountered a trade-off between supporting advanced features and making common features simple, we always tried to give priority to simplicity. MOOSE is tuned to make the set of things that are natural to do in a MUD—like bears that you can tickle—as easy as possible.

### 3.2.8 Hide Nasty Things Under the Bed

Perhaps one of the most-discussed issues in programming language design is how much detail to hide from the user. Maximum efficiency dictates that the user take direct control of as many functions as possible. For example, C requires users to allocate and deallocate memory manually. Usability usually dictates that the system do as much as possible of the unpleasant and error-prone parts of the task. For example, Lisp and MOO do automatic garbage collection. It should come as no surprise that for MOOSE, we chose the second approach, hiding difficult concepts where possible. In particular, we succeeded in hiding issues of permissions, atomicity and multitasking.

#### Permissions

Security for multi-user environments are an important topic of current computer science research. Over time, through intensive use and frequent attacks, the LambdaMOO server and LambdaCore[6] database have been made robustly secure. The permissions system that has evolved, however, is the most difficult part of MOO to master.

One particularly thorny issue concerns ownership of properties. Objects have owners, and individual properties also have owners. Properties have permission bits r (readable), w (writeable, which is almost never used), and c (change owner on inheritance). Suppose that Ginny owns Generic Dog, and Amy has a child of Generic Dog named Pumpernickel. Generic Dog has a bark_msg property. Now suppose Generic Dog has a "guard" script that changes the dog's bark_msg from a polite yip to a deep growl. If the bark_msg property is set +c (i.e. with the c bit set), then Amy will own Pumpernickel's bark_msg property, and can change it directly. However, the "guard" script runs with Ginny's permissions, so now it can't change the dog's bark_msg. If the property is !c (i.e. with the c bit unset), then the "guard" script can change the dog's bark, but Amy can't. The solution is to make it !c, and have Ginny write a change_bark accessor script allowing dog owners to change their dogs'

[6]To start a MOO, you need both the server and an initial database. It's possible to start with a minimal database that adds little or no functionality. However, most MOOs use a database extracted from LambdaMOO (the first and still most popular MOO) called LambdaCore. Among the features included in LambdaCore is the line editor discussed here.

barks. If you’re confused, that’s the point—permissions are difficult to understand.

In MOOSE, I found a way around this problem. I added a trust_parents property to all objects. If an object trusts its parents, then any code on an object’s parents may modify any of its properties. All MOOSE properties are +c—if you own the object, you own the property. The concept of the c bit is eliminated. Generic_Dog’s growl script can modify Pumpernickel’s bark_msg because Pumpernickel trusts her parents. (This does not mean that other code by Ginny can modify anything about Pumpernickel—just code on the parent object.) This involves assuming some trust between the owner of a generic and the owner of the child objects. However, that trust already exists. The generic owner can already add any property or script he or she likes to the parent, which will be inherited by all child objects. The generic owner would also be able to modify properties in the common MOO solution to this problem described above. MOOSE lets you do the things that it seems natural to do—have both Generic Dog and Pumpernickel’s owner be able to modify Pumpernickel’s bark_msg. It’s not necessary for the kids to think about property permissions at all.

The w (writeable) bit is not supported either. There is no good reason in MOOSE or MOO to make anything world-writeable. The r (readable) bit for properties is a more difficult issue. It’s beneficial to the community if scripts and properties can be used as examples by others in the community. On the other hand, making things unreadable is useful for applications like secret passwords, which children often are interested in writing. I was initially hesitant to introduce the notion of a property permission at all. However, after some discussion between Austina Vainius and myself and the independent suggestion of the same idea by Pavel Curtis, we finally compromised on adding the notion of a “hidden” property. We called them “hidden” rather than “secret” to make them sound less exciting. It’s desirable to minimize their use, so that people can learn from one another’s projects as much as possible.

**Atomicity and Multitasking**

In any time-sharing computing system, you need a system for deciding how to share processor time. MOO uses single-threaded, non-preemptive multitasking. Each task has a limit on the number of ticks and seconds it can use. If it runs out, the task stops with an out-of-ticks or out-of-seconds error. Before a task runs out of time, the programmer can voluntarily give up control using the “suspend” command. When the task returns to the top of the queue, it has refreshed tick and seconds counts (half as many as it originally started with).

I wasn’t particularly in the mood to teach children about tick limits and suspending. Furthermore, MOOSE adds sufficient overhead to the server that even simple tasks often must suspend at least once before completing. I decided to make MOOSE automatically suspend when needed.

This model has drawbacks. If you are petting a dog, you can no longer assume that a dog that was in the room at line 1 is still there at line 10! There is no way to guarantee that any task completes atomically. In practice, this has not yet caused significant problems. A few kids have noticed that messages generated by objects entering a room sometimes arrive in an odd order. For example, my dog Pumpernickel wags her tail when she enters a room containing someone who has pet her in the past. The message that she wags her tail often precedes the message notifying you that she has arrived in the room. Each command in an individual program is guaranteed to be executed in the correct order; however, the arrival message and the tail wagging are generated by different programs on different objects. The order of turn-taking between programs running on different objects is not guaranteed. While this simple example can be passed off as an oddity, the problems are likely to become more severe as the system grows in complexity.

As Mitchel Resnick's research on StarLogo has shown, most people are not generally comfortable thinking about complex, parallel systems. StarLogo seeks to make that complexity understandable and interesting in itself (Resnick 1994). I chose not to make understanding these issues a pedagogical goal for MOOSE. Instead, I decided to hide them from the user as much as possible. This works for limited applications, and has helped make the system accessible to novice programmers. However, this simplistic solution has limits. More research is needed to devise a gracefully scaleable solution to meet the needs of both novices and experts, and to enable the construction of more complex systems.

### 3.2.9 A Design Philosophy

There are few "right" answers in programming language design—primarily, there are trade-offs. Furthermore, it's not particularly meaningful to talk about one language being "better" than another. However, it is meaningful to talk about the advantages of a language for a particular group of people with a particular set of goals. And languages do have affordances—things they make easy to express, and things they make hard to express. Larry Wall, inventor of the Perl language, compares different programming languages to different styles of music—C is reductive and rigid like the Modernism of John Cage; "C++ is like movie music, of titanic proportions, yet still culturally derivative"(Wall 1996). Wall argues that programmers are like artists, and different languages are like different types of materials—they are expressive in different ways, and suited to people with different goals and personalities.

Our goal in designing the MOOSE language was to give children a new expressive medium. Towards this end, we tried to make it as easy as possible for children to write the sort of programs one tends to want to write in a

MUD. Our general approach was deliberately a bit mischievous, exploring a design aesthetic that is counter-intuitive for most computer scientists: put simplicity and immediate intuitiveness first, and ignore as much as possible many of the concerns that adult computer scientists tend to value. We tried to ground the design in experience working with real children with Logo, and revised the design based on feedback from initial users. Detailed analysis of what children have been able to accomplish with MOOSE forms the subject of much of the rest of this thesis.

## 3.3 The Need for a New Programming Environment

While existing MUD languages present a barrier to children learning to program, available programming environments are an even bigger problem. In MUSE, every command is one-line long, so no editor is necessary. Programming in MOO requires the use of an editor. The “@program” command used to enter programs is built into the server software. To modify your program using just that command, you would need to type the entire thing again. A very nice Emacs-based editor, mud.el, is available. Mud.el gives you an excellent interface to edit both MOO verbs and properties. However, not everyone has access to Emacs, and it’s certainly not appropriate for children. An alternative, currently used by the Pueblo project, is the LambdaCore line editor.

Here is the help for editor commands:

```
Verb Editor
Commands:

say          <text>                      w*hat
emote        <text>                      e*dit       <obj>:<verb>
lis*t        [<range>] [nonum]           com*pile    [as <obj>:<verb>]
ins*ert      [<ins>] ["<text>]           abort
n*ext,p*rev  [n] ["<text>]               q*uit,done,pause
enter
del*ete      [<range>]
f*ind        /<str>[/[c][<range>]]
s*ubst       /<str1>/<str2>[/[g][c][<range>]]
m*ove,c*opy  [<range>] to <ins>
join*l       [<range>]
fill         [<range>] [@<col>]

----  Do `help <cmdname>' for help with a given command.  ----

<ins> ::= $ (the end) | [^]n (above line n) | _n (below line n) |
   .(current)
<range> ::= <lin> | <lin>-<lin> | from <lin> | to <lin> | from <lin>
   to <lin>
<lin> ::= n | [n]$ (n from the end) | [n]_ (n before .) | [n]^ (n
   after .)
`help insert' and `help ranges' describe these in detail
```

Here's how the program to pet Rover (discussed in Section 3.1) would be created and entered using the LambdaCore line editor:

```
>@verb rover:pet this none none
Verb added (0).
>@edit rover:pet
Verb Editor
Do a 'look' to get the list of commands, or 'help' for assistance.

Now editing #2156:pet.
>enter
[Type lines of input; use `.' to end or `@abort' to abort the
   command.]
>player:tell("You pet Rover.");
>this.location:announce_all("Rover wags his tail.");
>.
Lines 1-2 added.
>compile
#2156:pet successfully compiled.
>q
Amy's Office
Amy's office is a jumble of books and papers.
```

Being able to break code up into multiple lines instead of cramming lines together as they are in MUSE is a significant improvement, but it comes at the price of adding this elaborate editor. To change "his" to "her" in MOO without a client, we would need to do this:

```
>@edit rover:pet
Verb Editor

Do a 'look' to get the list of commands, or 'help' for assistance.

Now editing #2156:pet.
>list
  1: player:tell("You pet Rover.");
__2_ this.location:announce_all("Rover wags his tail.");
^^^^
>s / his/ her/2[7]
__2_ this.location:announce_all("Rover wags her tail.");
>compile
#2156:pet successfully compiled.
>q
Amy's Office
Amy's office is a jumble of books and papers.
```

Perhaps the worst feature of the editor is that it actually moves you to a separate room to do your coding. The rationale for this design decision was to allow reuse of the "say" command for inserting text, and give you access to a simple set of editing commands. Moving you to a separate room was the

[7]Note the spaces before "his" and "her." The first time I typed this while writing this chapter, I actually forgot the space and ended up changing "this" to "ther." Even for an experienced user, this kind of editing is frustrating.

simplest way of achieving a form of modal interface (Curtis 1996). The downside of this design decision is clear. One of the great strengths of MUDs is their collaborative nature. Using the LambdaCore line editor, you are sent off to a room all alone whenever you try to work! (The collaborative nature of learning in MUDs will be discussed in more detail in Chapters 4 and 5.)

## 3.4 The Design of the MacMOOSE Client

There were few if any research issues involved in designing the MacMOOSE client program—just attention to detail, interface design work, and an iterative design process incorporating feedback from children. While I implemented the MOOSE language, code for the MacMOOSE client was written by several MIT undergraduates, participating in the project through MIT's Undergraduate Research Opportunities Program (UROP). The interface design was done by myself and the students jointly. The students who worked on the project are listed in Table 3.6.

| | |
|---|---|
| Greg Hudson | September 1992—May 1993 |
| Adam Skwersky | June 1993—May 1994 |
| Steven Tamm | February 1994—May 1995, and September 1995—May 1996 |
| Jon Heiner | February 1995—May 1996 |
| Drew Samnick | January 1996—May 1997 |
| Steven Shapiro (Java version) | September 1996—present |

Table 3.6: The MacMOOSE Development Team

MacMOOSE allows you to:

- Edit properties and code, and send MOOmail in Macintosh (WYSIWYG) style,
- Open an "object browser" to see all the scripts, verbs, and properties on an object, and
- View help in a separate window.

The application was designed to make it as easy as possible to learn to program, and to do creative writing. Features that were suggested by users or members of the development team that were excluded as not contributing to this goal include for example: automatic mapping, a separate window displaying who is currently connected, and a separate window showing what you are holding. Suggested features that were not implemented due to lack of time but which would advance the project's primary goals include: flashing matching parentheses, automatically coloring keywords in code, search and replace, support for alternate character sets (for students using MacMOOSE in language classes), and hypertext help (allowing you to double-click on a word in order to get help on that topic). Much of the time spent in design meetings was devoted to fighting “featuritis,” in order both to keep the interface easy to use and to keep the development time manageable.

We chose to develop MacMOOSE in Symantec C, because that was the best available development platform in September 1992. Between MacMOOSE version 1.0b1 and version 2.0a1 we moved to Symantec C++, which enabled more code reuse and better design of class abstractions. There was no adequate platform-independent development environment available at the time the project was started. A significant disadvantage of using the MacMOOSE client program is that it requires a Macintosh with a direct Internet connection. Had Java existed when we began its development, we would have written it in Java to reach a broader variety of platforms. A Java version of MacMOOSE is currently under development. While Java runs on multiple platforms, those platforms all require direct Internet connections as well. The LambdaCore line editor will run on anything—even a dumb terminal connecting over a phone line. The greater functionality of MacMOOSE is achieved at the cost of platform dependence.

### 3.4.1 A Tour of MacMOOSE

Connecting to MacMOOSE, you first select a server from a list (Figure 3.2). You can have multiple connections open at once. Each connection is identified by a number, and you can switch between them using command keys. Windows associated with a particular connection are grouped together on the windows menu. (These features are more necessary for adult MOO programmers than for children using MOOSE. As well as supporting children programming on MOOSE Crossing, MacMOOSE also works as a general-purpose MOO client. As of June 1996, more than 950 people had registered copies.)

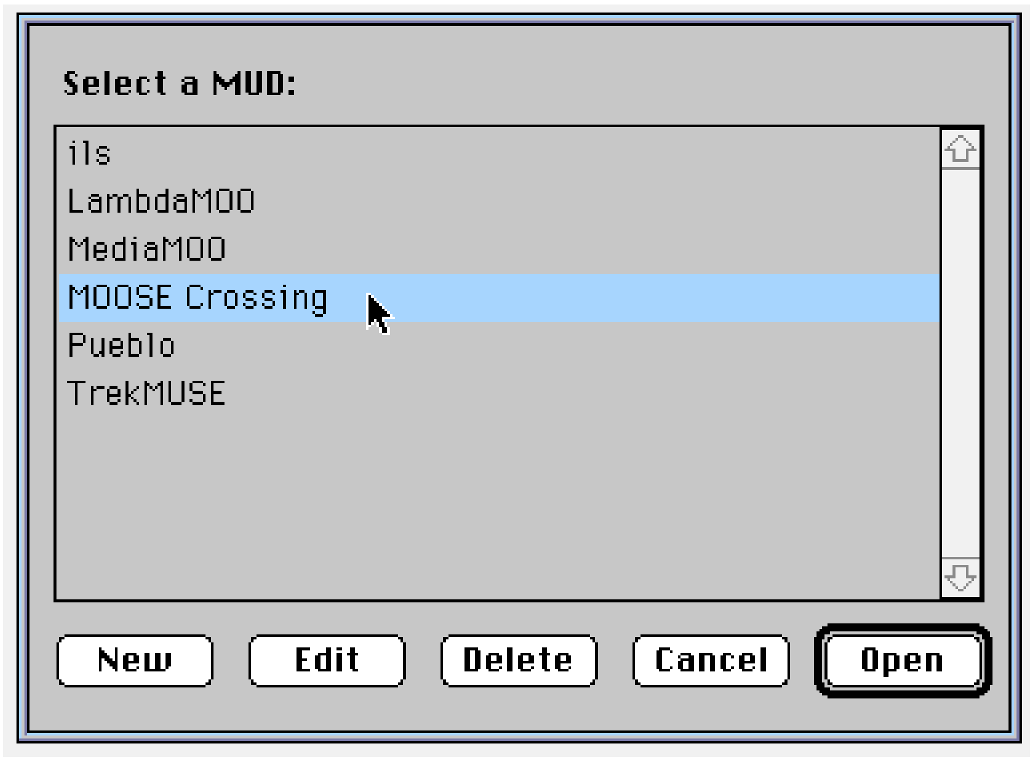
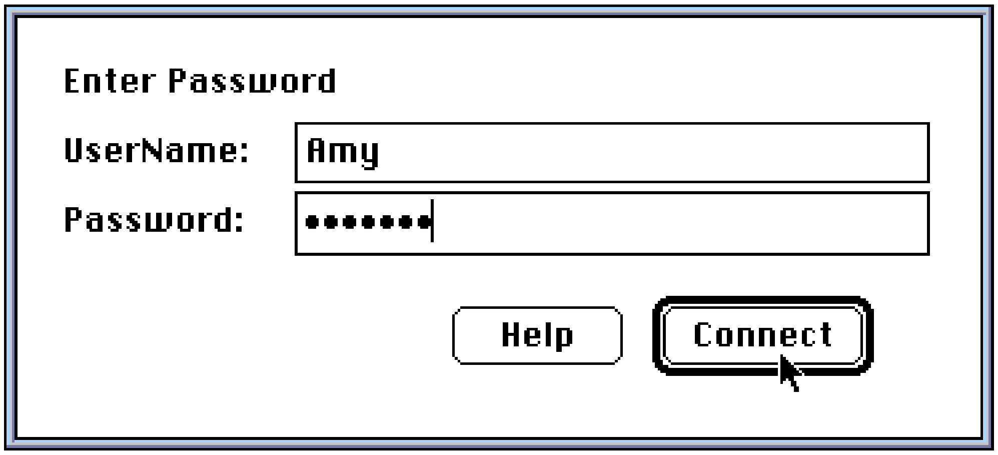


Figure 3.2:  The MacMOOSE Server List

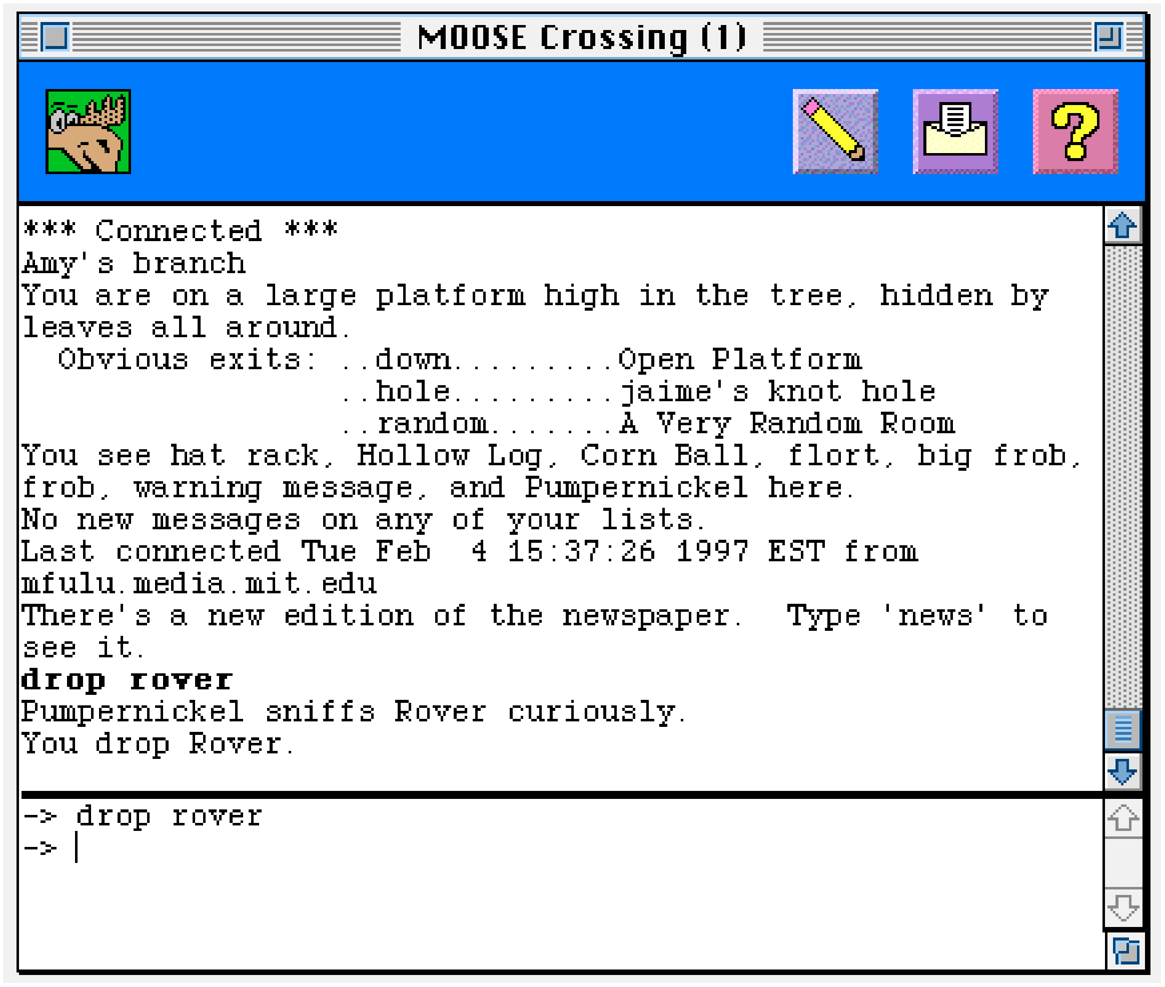
A password dialog lets you enter your password privately (Figure 3.3).  With most other MUD clients, your password echoes in clear text.  We worried this might lead to some children pulling pranks after they had seen each other's passwords.

Once connected, your main connection window is divided into two panes: input and output (Figure 3.4).  In a raw telnet connection, incoming text can make it difficult to read the line you're in the middle of typing.  The most common reason people use MUD clients is to solve this interface problem.



|   |   |
|---|---|
| UserName: | Amy |
| Password: | ••••••• |

Figure 3.3: The MacMOOSE Password Dialog



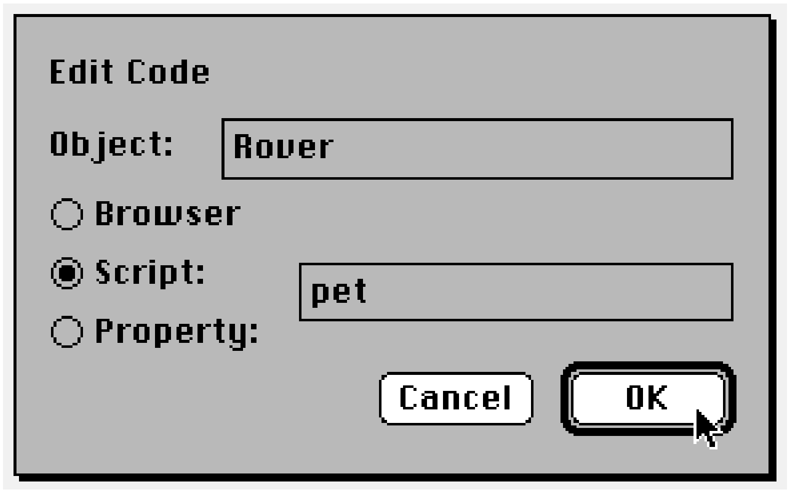
```
*** Connected ***
Amy's branch
You are on a large platform high in the tree, hidden by
leaves all around.
  Obvious exits: ..down.........Open Platform
                 ..hole.........jaime's knot hole
                 ..random.......A Very Random Room
You see hat rack, Hollow Log, Corn Ball, flort, big frob,
frob, warning message, and Pumpernickel here.
No new messages on any of your lists.
Last connected Tue Feb  4 15:37:26 1997 EST from
mfulu.media.mit.edu
There's a new edition of the newspaper.  Type 'news' to
see it.
drop rover
Pumpernickel sniffs Rover curiously.
You drop Rover.
```

```
-> drop rover
->
```

Figure 3.4: The MacMOOSE Main Connection Window

What is more unusual about MacMOOSE compared to other MUD clients is its emphasis on supporting programming. Consider the program we discussed before: petting Rover. Here’s how you would write the same program using the MacMOOSE client program. First, you’d click on the pencil icon (or select “Edit...” from the MOOSE menu) to indicate that you want to edit something.

Next, you’d enter the name of the object and the script you want to edit in the dialog box that appears (Figure 3.5).


Edit Code

Object: Rover

○ Browser

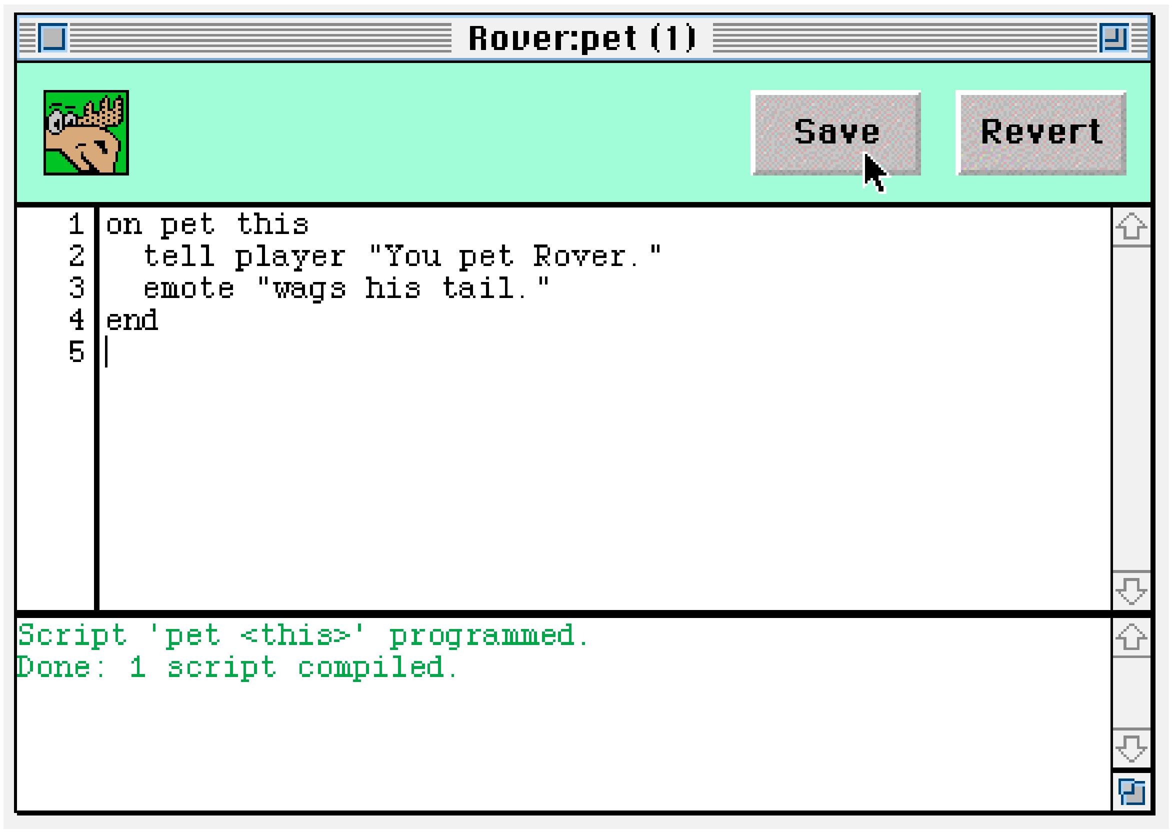
[✓ ● Script:] pet

○ Property:

Figure 3.5: The MacMOOSE Edit Code Dialog Box

MacMOOSE asks if you want to add that script. Instead of having to remember a special command to declare a new script, the client does it for you. When you click OK, you get an editor, in which you can simply type your program (Figure 3.6).

In the script editor, you can change text in Macintosh WYSIWYG (What You See Is What You Get) style. If you wanted to change “his” to “her,” you’d just click after “his,” hit delete twice, and then type “er.” Children are able to figure this out with little or no help. In version 1.0a1, feedback from compiling a script or verb appeared in the main window. This was awkward because you are not looking at the main window when you compile code. In version 1.0b1, we added a feedback area at the bottom of the editor windows, so your feedback is closely associated with your code. Screen real estate proved to be tight—you need adequate space to see the feedback but also adequate space left to edit code. The MOOSE compiler runs on a remote machine; the client runs on a local machine. We tried to minimize that gap by integrating compiler feedback with the client interface.



```
on pet this
  tell player "You pet Rover."
  emote "wags his tail."
end
```

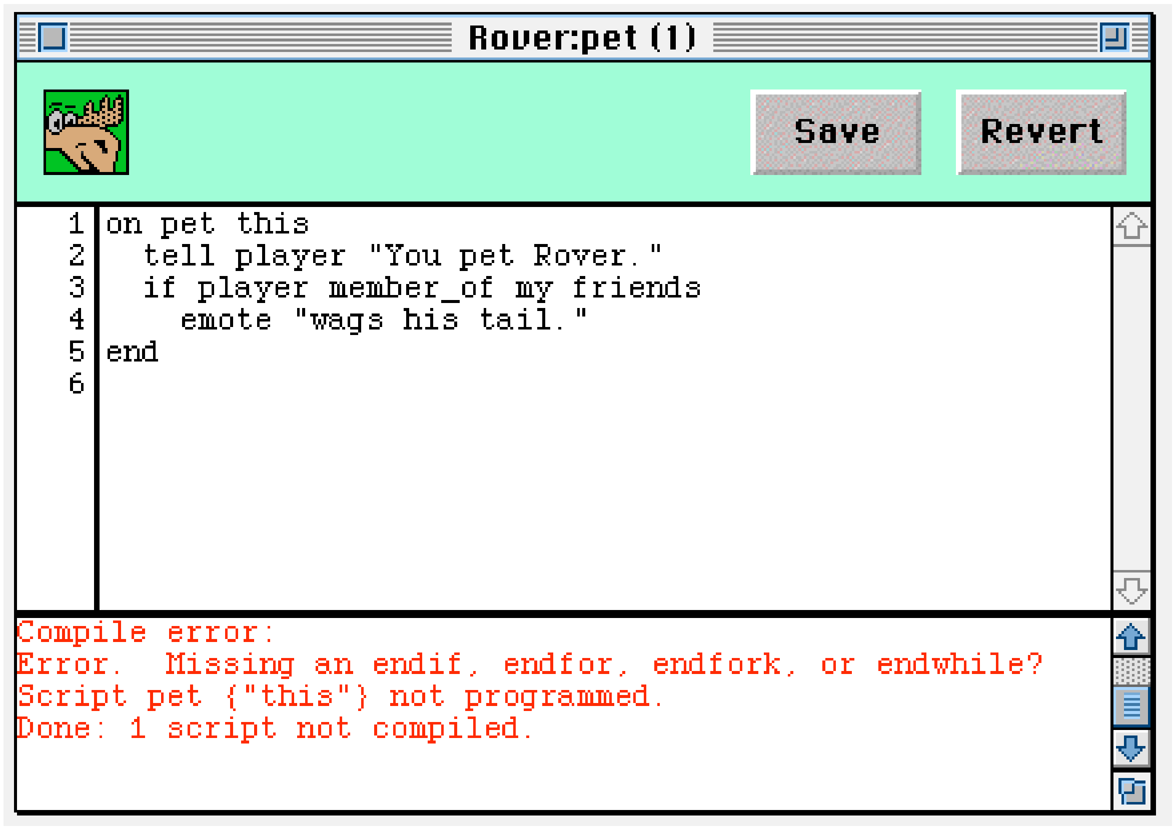
```
Script 'pet <this>' programmed.
Done: 1 script compiled.
```

Figure 3.6: The MacMOOSE Script Editor

Unfortunately, we found that children often did not look at the feedback. Version 2.0 made positive feedback green and negative feedback red (Figure 3.7). Your most recent feedback is colored; feedback from previous compiles is turned black. While the children still may not always read the feedback, they know that red means something is wrong, and they notice the red text appearing. Additionally, the MOOSE icon[8] at the top of the window is turned red while the transaction is in progress and green again when it has completed. This gives users a good indication of when their compile is done. (Compiling a short script is immediate, but long ones can take several seconds.)

To edit an object, you can open up a browser on that object (Figure 3.8).

[8]The moose icon was designed by Michael Maier for MacMOOSE.

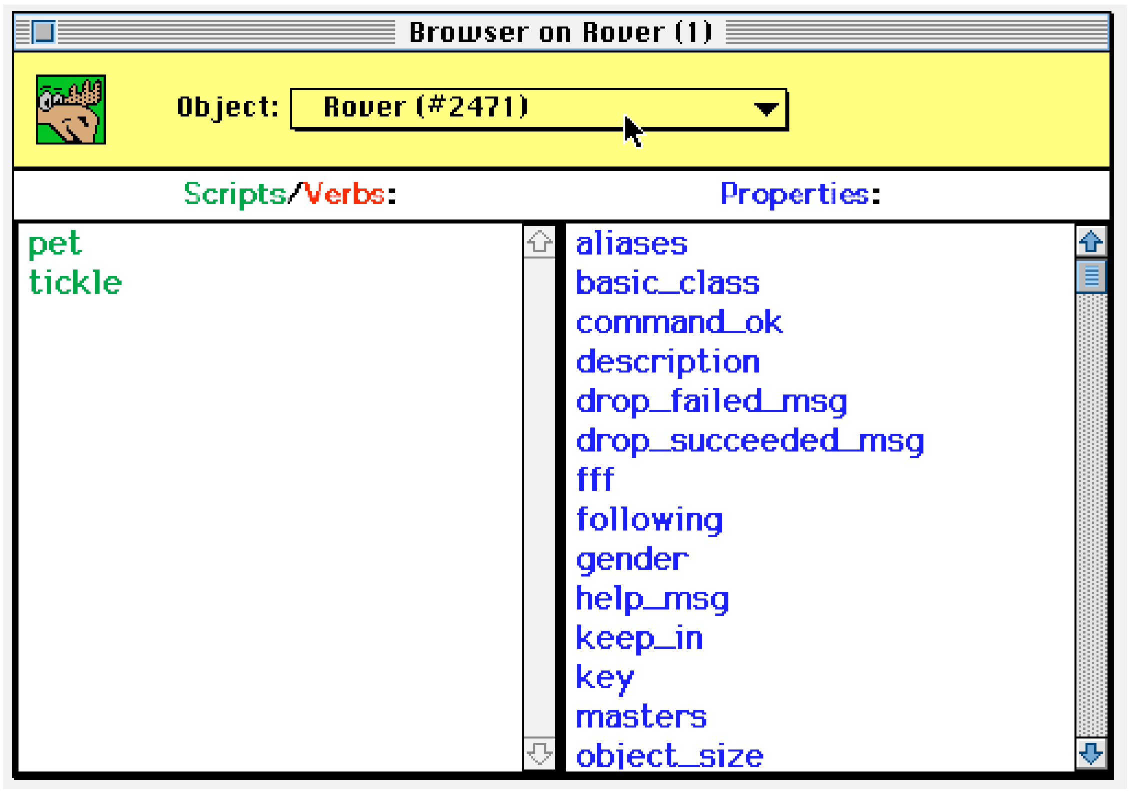


```
on pet this
  tell player "You pet Rover."
  if player member_of my friends
    emote "wags his tail."
end
```

```
Compile error:
Error. Missing an endif, endfor, endfork, or endwhile?
Script pet {"this"} not programmed.
Done: 1 script not compiled.
```

Figure 3.7: Feedback for a Compile Error

The central pop-up menu lets you climb the inheritance hierarchy to edit the object's parents. All of the object's scripts are shown on the left, and properties on the right. Double-clicking on a script or property opens an editor for that script or property. You may notice that fewer scripts are listed than properties. Script and property inheritance are handled slightly differently. If a parent object has a property, all of its children can have their own values for that property. If a parent object has a script, all of its child objects may use that script as it exists on the parent, but they can't have their own version of the script. For this reason, the only scripts listed are those declared on that object; properties listed include those declared on that object and on all of its parents. Adam Skwersky and I struggled with this interface design issue for over a month. The final solution significantly helps users work with these subtle differences in inheritance easily.



pet
tickle

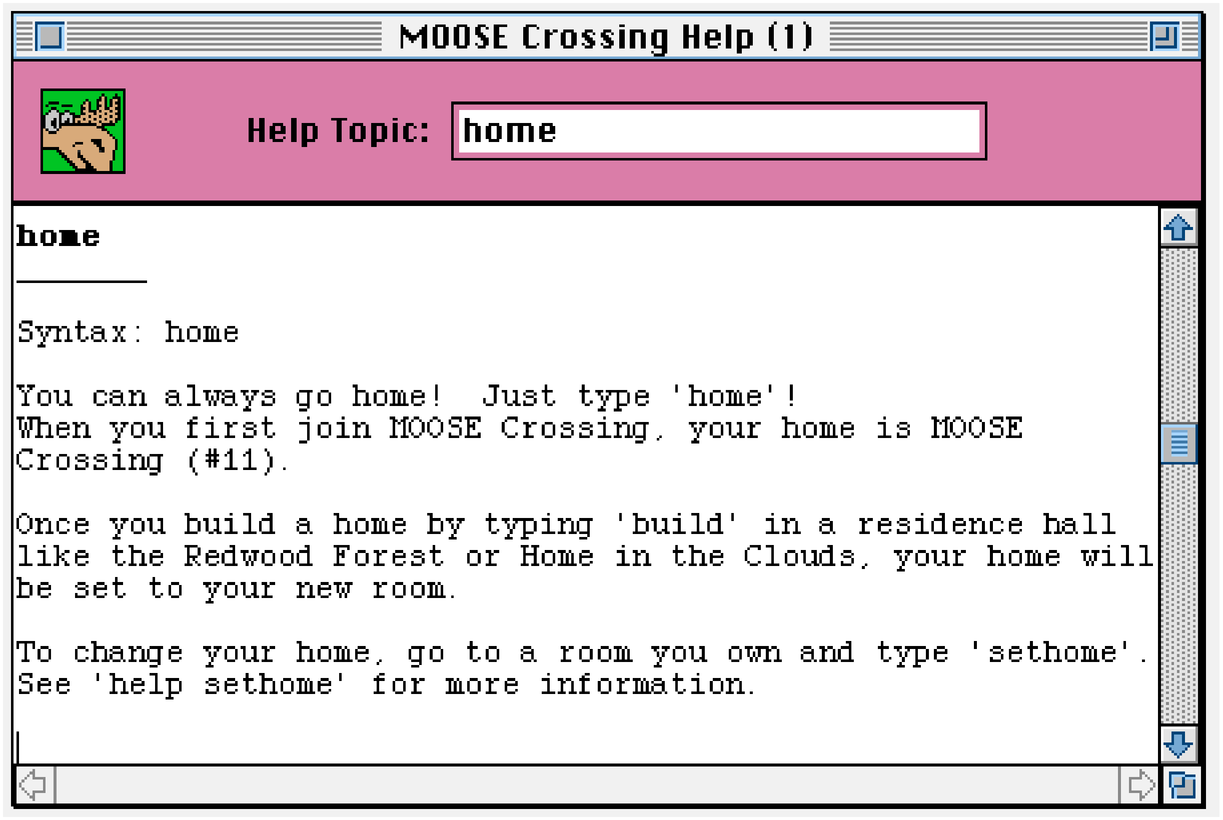
aliases
basic_class
command_ok
description
drop_failed_msg
drop_succeeded_msg
fff
following
gender
help_msg
keep_in
key
masters
object_size

Figure 3.8: The MacMOOSE Object Browser

If you need help, you can get that help in a separate window (Figure 3.9). Multiple help messages can go in the same 'help browser'. This prevents help messages from scrolling away as you try to do what they recommend.

Finally, we added an interface to allow users to send MOOmail (mail internal to the virtual world) with WYSIWYG editing (Figure 3.10). The interface allows for sending mail, but not more advanced features like replying to a message including part of the previous message in the body.[9]

[9]The MacMOOSE documentation quips "no, we are not writing MOOdora."

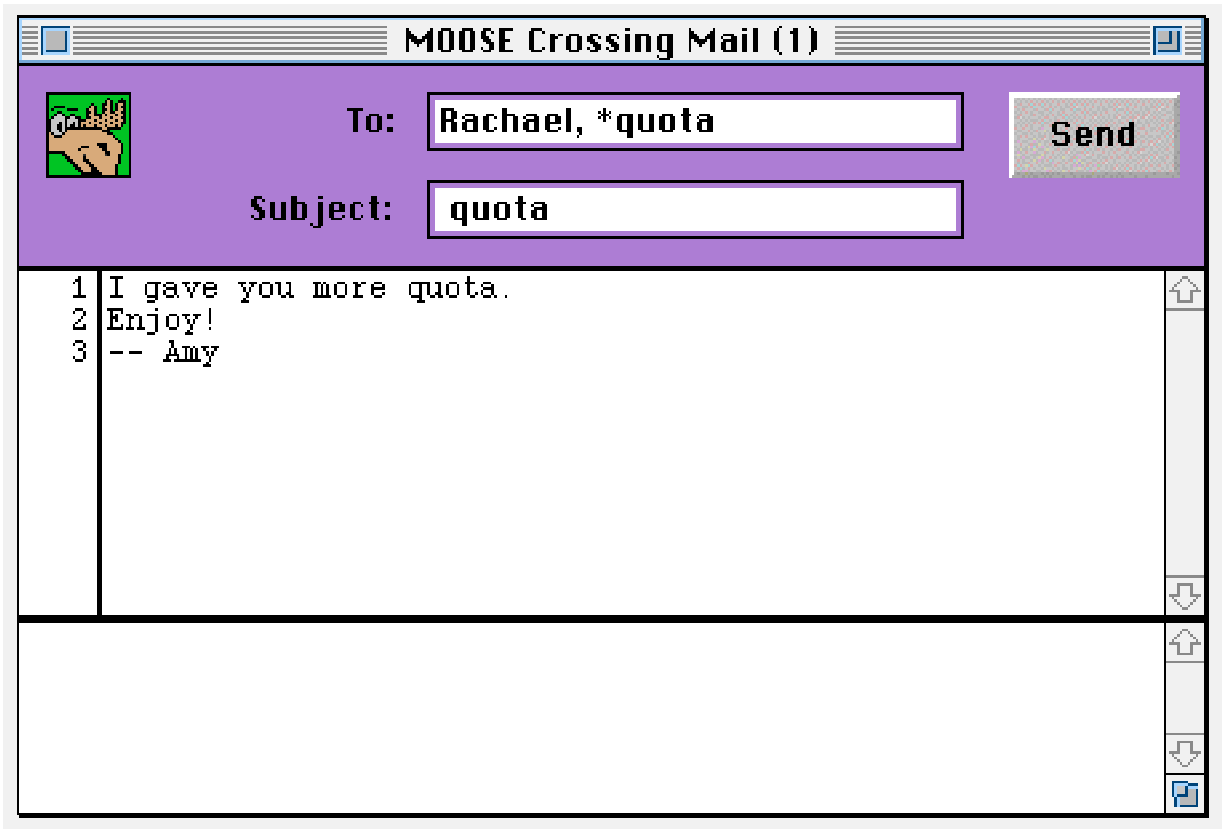


```
home
______

Syntax: home

You can always go home!  Just type 'home'!
When you first join MOOSE Crossing, your home is MOOSE
Crossing (#11).

Once you build a home by typing 'build' in a residence hall
like the Redwood Forest or Home in the Clouds, your home will
be set to your new room.

To change your home, go to a room you own and type 'sethome'.
See 'help sethome' for more information.
```

Figure 3.9: The MacMOOSE Help Browser

### 3.4.2 Equal Access for Few Versus Unequal Access for Many

MacMOOSE has proved to make learning to program significantly easier for children (and adults). While it is possible to access MOOSE Crossing without MacMOOSE, those doing so are at a significant disadvantage. Not wanting to create a class of haves and have-nots, I chose to restrict access to MOOSE Crossing to those who have access to Macintoshes on the Internet and can use MacMOOSE. This has unfortunately significantly limited the number of kids who have been able to use MOOSE Crossing. The number of children with access to Macintoshes is a small proportion of the total. Giving all participants equal access has made access available to a much more restricted group of children. That group is unfortunately disproportionately wealthy. We have been actively working with organizations such as CTCNet and PluggedIn who provide computing facilities to less advantaged children to try to broaden the demographic of children who have access to MOOSE Crossing. We hope to begin work on a Java version of MacMOOSE in the near future, which should make MOOSE Crossing accessible to a larger number of children.



Rachael, *quota

quota

```
I gave you more quota.
Enjoy!
-- Amy
```

Figure 3.10: The MacMOOSE Mail Interface

## 3.5 Designing Empowering Technologies

Technology increasingly surrounds our everyday lives—the World Wide Web was only invented in the early 1990s, but by 1995 there were already URLs on bus ads. To what extent will the general public have meaningful control over those technologies? The answer is not clear. I believe that if you give people quality tools and social support for the use of those tools, they will surprise you with their intelligence and creativity. Part of taking users seriously involves including them in all stages of the design process: grounding design decisions in observations of real users rather than the formal concerns of professionals, and revising designs based on feedback from initial users. Users rise (or fall) to designers' expectations. We began the design of the MOOSE language and the MacMOOSE client with the assumption that kids are capable of great things. Our design agenda was also a political agenda—technology can and should empower people. I believe designing for nonprofessional users is a central issue for the future of computer science.

# 4. Collaborative Learning Strategies: Storm's Weekend with Rachael

## 4.1 Storm's Weekend with Rachael

One Friday afternoon in April of 1996, I accepted the MOOSE Crossing application of a new member, a twelve-year-old girl who chose the character name Storm, and then I left town for the weekend. I usually log on periodically over weekends, but this particular weekend I went to Maine and was offline. When I returned on Sunday, I was surprised to learn that Storm now knew the basics of how to program. She had had limited previous experience—she had once tried Logo in school. Over the weekend, she made Jasper (a frog you can hug), Callie (a cat who purrs when you tickle her and responds wryly when you address her as "cat" rather than by name), and a catnip mouse for Callie. She also built and described three homes, and collaborated with Rachael (girl, age 13) on an extension to Rachael's castle, complete with a dead princess on the floor who had died for love. Storm and Rachael spent most of the weekend together, talking and helping one another with their projects.

I've told this story many times, and people often shake their heads with disbelief, grinning. But what is there about the story that is so remarkable? It's certainly not how we stereotypically imagine two children spending a holiday weekend. It's hard for many people to imagine any "real people" having meaningful control over computational media, much less girls. What was it that interested them so completely that they would spend most of the weekend absorbed in it? What were they doing and how did they learn to do it?

Another set of questions concerns the nature of the collaboration. What prompted Rachael to spend an entire weekend helping a stranger? This is quite different from a group of students being assigned in school to do a group project. Rachael chose to help Storm for an extended period of time simply because she enjoyed doing so. What began at the start of the weekend as an expert assisting a novice quickly became a joint effort in which what is achieved is greater than what either participant could have accomplished individually. Their collaboration was the beginning of a friendship.

In this chapter, I describe in detail what took place over that weekend. Everything typed on MOOSE Crossing is recorded, with written informed consent from both parents and children. Over that weekend alone, 3.7 Mb of data was recorded of their experiences. This data includes everything each girl typed and saw on the screen—everything they did, and everything they said to one another online. By examining what took place in extended detail, I am attempting to present what Clifford Geertz calls a "thick description" (Geertz 1973). Methodologically, studying this online medium has interesting advantages: we have a complete record of the interaction between these two girls. They could communicate only through the computer, and every keystroke was recorded.

Seven months after this weekend took place, I sent Storm and Rachael copies of an earlier version of this chapter to read and comment on. Social science has in recent years increasingly rethought the issue of ethnographic authority. Henry Jenkins writes that "The newer ethnography offers accounts in which participation is as important as observation, the boundary between ethnographer and community dissolves, and community members may actively challenge the account offered of their experience" (Jenkins 1992). I chose to give the girls an opportunity to respond to my account of their experiences. After they had each read the chapter, I invited them to come to the Media Lab to discuss it with me, and to meet one another for the first time. It was fortunate that Rachael lives in the greater Boston area, and Storm lives an approximately ninety-minute drive away. (I'm grateful to Storm's parents for taking the time to make the trip.) I interviewed them both individually and together about their experiences that weekend and on MOOSE Crossing in general, and recorded those interviews on audio tape. Over all, they enthusiastically concurred with my account. As I had hoped, the interviews helped me to clarify what they each were thinking and feeling over the course of the weekend. As an added bonus, it also gave me an opportunity to ask the girls in what ways meeting face to face was different from meeting online.

There are four primary goals to this analysis:

- to give the reader a more concrete feel for what participating in MOOSE Crossing is like,
- to explore what children find compelling about this environment,
- to explore the learning strategies children typically use on MOOSE Crossing, and
- to explore a style of collaboration that often occurs there.

In my first outline for this thesis, there was a chapter called "Learning Strategies" followed by a chapter entitled "Collaboration and Learning from Peers." As soon as I sat down actually to write, it became clear that these were one chapter. My error was telling—we tend to think of learning first, and collaborative learning as a special sub-topic. In fact, all learning takes place in a social context. As a result, almost all learning has collaborative aspects. Even when people undertake learning experiences alone, they are still in some way responding to a broader social context—to the expectations of others, and the way we define our sense of self through our accomplishments and in relationship to others. MOOSE Crossing was designed to support the sort of self-directed, peer-supported, collaborative learning that took place for Storm and Rachael over this weekend.

## 4.2 Friday: Diving In

Rachael, who at this time had just turned 13 the previous week, is one of MOOSE Crossing's most dedicated and accomplished regulars. She was the first home-schooled student to join. Being home-schooled gives her greater time to devote to MOOSE Crossing, and also greater need for social contact.[1] An hour and a half after I accepted Storm's application, Rachael checks the list of all members, notices that there is a new member who hadn't logged on yet, and sends her this mail:

```
Message 1 on Storm:
From:     Rachael
To:       Storm
Subject:  hi

Dear Storm,
Hi! My name is Rachael. Who are you? I am thirteen years old and I am
female. I have been on moose crossing scince january and whould love
to be your friend.  The best times to go on moose crossing are on
mondays and fridays afterschool.

Rachael
```

At a little before 5 PM, Storm connects for the first time. Her connection message tells her she had mail, so she checks "help mail" and figures out how to read her message from Rachael. She told me in a later interview that she learned how to use the online help system and a few other basic commands from the introductory message mailed to all new members with their passwords. She learned about the mail system because she wanted to read her mail from Rachael—interactions with other children were an integral part of her explorations with the system from the very first command typed.

A moment later, Rachael notices that Storm has connected, and pages her "May I join you?" ("Paging" is a way of communicating with someone who is not in the same room in the virtual world.) Rachael waits impatiently for Storm to respond, repeatedly checking 'who' (a command that tells you who is logged on, how long they've been connected, and how long they've been idle). She also repeatedly checks Storm's last commands. It's possible to see all the commands someone has typed recently. We added this feature to MOOSE Crossing to make it easier to help others figure out what they're doing wrong when things aren't working right. When you look at someone's last commands, the system tells the person that you have looked. Thirty seconds later, Storm pages Rachael "Yes" and Rachael joins her, moving to be in the same room as Storm in the virtual world:

[1]Over time, MOOSE Crossing has become particularly popular with home-schoolers. The open-ended, self-directed educational philosophy behind MOOSE Crossing is consistent with the educational philosophy of most of the home-schooling movement. Additionally, it provides much-needed social contact with peers.

```
Rally says, 'Greetings Clover'
Rally arrives, following Rachael.[2]
Rachael says, 'hi'
Storm says, 'hello, all'
Rachael smiles.
Rachael says, 'Rally and Clover are my pets.'
Storm smile
Rachael says, 'how old are you?'
```

[Here Rachael looks at Storm; her description is still blank.]

```
Storm says, '12'
Rachael nods.
Rachael says, 'Are you at the media lab or somewhere else?'
Storm says, 'this is the first time I've been here!:)'
Rachael says, 'I mean in real life where are you? I'm at my house.'
Storm says, 'same here'
Storm says, ' anyone here like star trek?'
Rachael says, 'Many people go to the media lab, in MIT, to do moose
   crossing. I was just wondering if you where. Yes, I do.'
Storm says, 'i'm a trekker, heart and soul!:)'
Rachael giggles.
Rachael says, 'austina/kristina[3] likes it too.'
Storm says, 'whoohoo!'
```

The expression “whoohoo!” has become popular over the last few years, and is taken from the television show *The Simpsons*. Homer Simpson says it when he's happy. Popular culture, particularly science fiction television, immediately gives these two girls something to talk about. Their mutual interest also identifies them to each other as being part of an unusual sub-group of teenagers: girls who like science fiction. While many cultural critics scorn popular culture as a negative influence on children (Postman 1985), in the right environment it can be a rich source of raw materials for personal expression (Jenkins 1992).

```
Rachael says, 'which generation do you like best?'
Rachael says, 'I like the TNG'
Storm says, 'trekkers will take over the world! I like the next
   generation
Rachael smiles. "Mabye so!"
Rachael says, 'same here, the next generation. Isn't a shame that it
   stopped?'
Storm says, 'oh, man yes! bummer!\'
Rachael says, 'Whould you like any help?'
Rachael says, 'I hear that the next TNG (next generation) movie is
   about the borg.'
```

[2]For technical reasons, events on MOOSE Crossing sometimes happen in an unusual order—Rachael’s pet Rally says hello before he is announced as arriving in the room.
[3]The characters Austina and Kristina are played by the same person, MIT undergraduate Austina Vainius, an undergraduate research assistant working on the MOOSE Crossing project.

Rachael guesses that Storm doesn’t understand that the abbreviation “TNG” stands for “The Next Generation,” and here finds a way to let her know without making a big deal out of it.

```
Storm says, 'oops, didn't mean the \. no, i'm figuring this out!
   thanks!'
Storm says, 'well, the movie is about the two generations meeting.
   You should see it!'
Rachael says, 'that's fine. Just askin'. But if you want any don't be
   afraid to ask.'
```

Storm hasn't yet had time to explore or get confused about anything yet—Rachael contacted her within moments of her arrival. Rachael's immediate offer of help is a bit premature. While Storm does not immediately avail herself of that help, she now knows that help is available when she needs it.

Rachael and Storm’s conversations about help with MOOSE Crossing and Star Trek become interwoven. This is typical of MUD exchanges. In face-to-face discourse, turn-taking conventions help to keep a conversation focused. In MUDs, while one person is responding to a previous point, another may introduce a new one. Typing in parallel often leads to many-threaded conversations (Cherny 1995).

```
Rachael says, 'Oh, I saw it. I meant the one they are making now.'
Storm says, 'anyone like monty python?'
Rachael says, 'Yes, but I never get a chance to see it.'
Rachael says, 'Have you heard of Babylon Five?'
```

Rachael explained earlier that Rally and Clover are her pets; however, from her use of “anyone,” it's not clear if Storm yet understands that it's only really Rachael there. At this point, Storm looks at Rachael. She sees:

```
A girl with brown hair and green eyes. On her head is a sliver
headband with silver strands. At the end of each strand is a silver
ball. Around her neck is a silver chain.
She is awake and looks alert.
Carrying:
Franky                                  Rachael's Bean
Rachael is wearing a tye die shirt and overalls.
Rachael smiles.
```

Rachael had written a special script (called a “look_self” script) which is run every time someone looks at her. She is now notified “Storm just looked at you.” The program also causes her automatically to smile.

Rachael next looks at herself, to see what Storm has seen. Storm tries to look at Franky and Rachael's bean, not realizing she can't see objects that someone else is holding. Rachael audits Storm (a command which shows you all the objects a person owns), and sees that she  still owns nothing.

```
Rachael says, 'Do you like the way I look?'
Storm says, ' you look bea-u-ti-ful!'
Rachael says, 'thanks.'
Rachael says, 'Well, have you heard of babylon five?'
Rachael says, 'It is another science fiction TV show.'
Rachael hugs Rally.
Rally squeals happily.
```

Rachael here gets a bit fidgety, typing “who” a couple times and hugging her pig Rally. Storm is busy describing herself. Rachael guesses as much, and looks at Storm immediately after the description is finished. She sees “you see a tall, black haired, white - skinned girl, wearing all black. she is wearing lots of silver jewelry.”

```
Storm says, 'never seen it'
Rachael says, 'Most people haven't. You look nice!'
[Storm here looks at Rally, and sees "A small pink piggy."]
Storm says, 'why thank you!'
```

Storm gets immediate positive feedback. The feedback has value on several levels. First, it's a compliment to her technical ability—she was able to figure out how to describe herself. Second, it's a compliment to her writing ability. Third, being told you look nice is a generally a mark of social acceptance. Finally, given the predominance of issues of appearance in the culture of teenage girls, being told that your virtual self looks nice potentially has deep resonances.

```
Storm says, 'monty python is very funny'
Rachael says, 'You can changer your gender by typing "gender me as
   <gender>"'
Rachael says, 'the <>s are where you fill in the blank.'
```

The MOOSE Crossing documentation is written in this style, with angle brackets indicating that you should fill in the blanks. A classic slapstick routine has a bailiff trying to swear in a witness by saying “Repeat after me. I, state your name.” The witness replies literally, “I, state your name.” Giving someone directions including some things to duplicate literally and others to interpret is awkward, and is a common source of confusion for new users on MOOSE Crossing. Rachael here is anticipating the problem, giving Storm useful information not just about one command but about how to read the documentation in general.

```
Rachael says, 'Yes, monty python is.'
Storm says, ' i know. I'll love doing that! what planet are you
   from?'
Rachael says, 'Depends.'
Storm says, 'i'm from zork'
Rachael says, 'Of what universe you are referring to-this one, star
   trek...my own..'
Rachael says, 'the game?'
Storm says, 'i'm referring to a computer game.'
```

```
   Storm says, 'yes. it's fun'
   Rachael says, 'I thought so. I've played it.'
   Rachael says, 'But I never passed it.'
   Rachael says, 'Have you played MYST?'
   Storm says, 'what one?'
   Rachael says, 'Zapies...I made it up myself when I was in 1st grade.'
   Storm says, 'no. my mom beat myst.'
   Rachael says, 'I haven't yet. It's kinda hard.'
   Rachael says, 'Do you live in the Boston area?'
```

Storm here attempted a bit of fantasy role playing. Rachael asked for a clarification—what is the reference frame of the conversation? They moved back to talking about real life.

```
   Storm says, 'could you show me around?'
   Rachael says, 'certainly!@'
   Rachael says, 'anywhere in particular?'
   Rachael says, 'or just everywhere?'
   Storm says, 'everywhere! please'
   Rachael says, 'well, let's go up!'
```

When Rachael first offered help, Storm wasn't yet ready to take advantage of it. Now that Storm has tried out a few basic commands and gotten to know Rachael a little bit, she's comfortable asking for help. Rachael leads Storm up to the clouds, where Rachael demonstrates how you can jump and roll around on one of the clouds. While Rachael is jumping around, Storm sets her gender to royal[4], making use of Rachael's earlier offered information.

Rachael is clearly delighted to meet Storm. She uses this command:

```
   --> tell Rally to emote hugs Storm.
```

This causes her pig to hug Storm. Storm hugs the pig back. The pig allows Rachael to express affection indirectly.

Rachael next leads Storm to Home in the Clouds, a room that lets you build your own home by simply typing "build." Storm builds a home there.

Many kids on MOOSE Crossing make use of a special character class that lets them switch between multiple identities. One of Rachael's alter-egos is called QueenAnne. Rachael morphs into QueenAnne and invites Storm to see her castle. Storm types "queenanne" to get to her room without being prompted, and Rachael follows. Storm bows to QueenAnne. QueenAnne changes back into Rachael, and leads Storm to Palladia, part of her mythical kingdom where you can build your own home. Rachael next explains how you can build a home in certain places (not realizing Storm has already done so at Home in the Clouds). Storm builds another home at Palladia. Rachael offers

[4]Changing your gender changes your pronouns. Having the gender "royal" means you are by default referred to by "the royal we."

to show Storm her “normal” home, but Storm replies she'd like to do the tutorial. Rachael goes home. Storm types “tutorial” and then starts looking at the dog tutorials, a three-part sequence that teaches the basics of MOOSE programming by helping you to make your own pet dog. Rachael pages good bye:

```
Rachael pages you.
She pages, 'I'm disconnecting now. I hope you come again. Mondays are
   the best days.'
```

Storm doesn’t reply. Rachael disconnects for about a minute, and then logs back on. Storm wanders into Rachael's room and looks at her pet penguin and cat. Rachael is there and says hi. Storm says hi back and goes out a different way than she came in. Rachael follows her. Catching up to her, she says “I can stay somewhat longer.” Storm leaves again, and Rachael hesitates and then followed her again, this time to the recycling center. Rachael uses the 'announce' command to print to the room “THE BLUE BIN EATS THE TRASH” followed by “CRUNCH” several times. (Printing unattributed messages like this to everyone in the room is technically against the MOOSE Crossing code of conduct. Members generally remind one another of the code of conduct when it gets out of hand, but tolerate it in moderation if the content stays friendly.) Storm says “I'm getting out of here.” She told me during a later interview that she didn’t realize Rachael had generated those messages. They walk to town together. At MOOSE Crossing, Storm says “hi! where to.” Rachael replies “You choose, I'll follow.” They arrive on Main Street:

```
Main Street
You're on the edge of Our Town. Looks like there's space to build
   some shops here!
Obvious exits: ..west.........MOOSE Crossing
               ..north........North Main Street
               ..east.........Town Hall
Storm is here.
Rachael has arrived.
Rally says, 'Hello Clover'
Rally arrives, following Rachael.
Clover arrives, following Rachael.
Rachael says, 'I suggest n'
```

[Rachael is suggesting they continue their explorations by going north. “n” is usually allowed as an abbreviation for north.]

```
Storm says, 'how do you make animals?'
Rachael says, 'Well, it depends if you want a new type of animal or
   one that already exists.'
Storm says, 'new type'
```

Here Rachael checks the command “parents Rally” and gets this response output:

```
Rally(#381)   generic_greeting_creature(#402)   Generic Teachable
   Object (#225)   Generic Puppet(#223)   Generic Following
```

```
      Object(#342)   Generic Gendered Object(#77)   generic thing(#5)
      Root Class(#1)

   Storm says, 'i'd like an animal to follow me around'
   Rachael says, 'type "create #223 named <name>".'
```

Storm does, and chooses the name Jasper. Rachael looks at Storm and sees she's now holding a creature. She tries to look at it, but Storm is holding it. She asks Storm to drop it. Storm describes it as “a frog” and drops it for Rachael to see.

```
   Rachael says, 'neat idea! I wish I thought of a frog!'
   Rachael says, 'to make it follow you type "set Jasper's following to
      me'
```

Rachael responds immediately with positive reinforcement, and additional technical advice. Storm improves Jasper's description to “frog with orange skin, a black stripe and a sense of humor.”

```
   Storm says, 'Jasper likes Star Trek too'
   Rachael smiles.
   Storm says, 'how do you make him say things'
   Rachael says, 'okay..now go somewhere, and it will follow you.'
   Rachael says, 'it depends.'
   Storm says, 'ok...'
   Rachael says, 'There are two ways. One, you could make a script, so
      that if you type something it will do something in return, like
      huging Rally..'
   Rachael hugs Rally.
   Rally squeals happily.
   Storm says, 'how'
   Rachael says, 'if you go to the pencil, it will make a thing appear.
      Fill in the blanks.'
```

[The pencil icon on MacMOOSE lets you edit an object, script, or property. See Figures 3.4-3.6]

```
   Rachael says, 'then, when you have a script ready to fill out, you
      type "on <script's name> this.'
   Rachael says, 'let me give you an example..'
```

Rachael lists out a script of hers, and says it out loud:

```
   Rachael says, 'here:on hug this    tell context "You hug " + my name
      + "."     announce_all_but context context's name + " hugs " + my
      name + "."     emote "squeals happily."   end'
   Rachael says, 'where the big blanks are, is where the returns are.'
```

Rachael waits a minute, repeatedly checking to see if the script is there yet.

```
   Storm says, 'what do you put on the pencil blanks'
   Rachael says, 'here, fill in the object as Rally, then fill in the
      script as "hug".'
   Rachael says, 'then you'll see what a normal script might look like.'
```

Storm first opens an editor on Rally:hug, and then on Jasper:hug. One of the most common learning strategies for kids on MOOSE Crossing is using other kids' programs as examples. In this respect, MOOSE Crossing shares a powerful learning feature with the World Wide Web. On the web, you can view the HTML code for any object. You don't need to go to a special library of examples—everything is an example. In the course of your normal use of the web you encounter objects that you can later go back to and use as models. Similarly, every object on MOOSE Crossing is an example. While the MOO language allows some programs to be unreadable to others, I deliberately chose not to support this feature. All programs can be viewed by everyone and learned from. Children often start with very simple variations on others' programs, and gradually progress to more original creations. This powerful learning strategy is often prohibited in schools, where it is declared to be "cheating."

It's worth noting that on MOOSE (unlike the web), you are likely to know or have the opportunity to meet the creator of each item in the world of examples. Storm is not learning from a randomly selected example; she's learning from an example created by her new friend, Rachael. The social and intellectual relationships are mutually supportive.

```
Rachael says, 'do you understand...or should I try explaining some
   more?'
Storm says, 'wait a sec..'
Rachael says, 'okay..i'll be right back..'
```

Rachael wanders around a bit and comes back to see how Storm is doing. Storm asks how she's supposed to save her work—whether she's supposed to click "Change" or "Revert". Rachael tells her to use 'Change'. (Based on feedback from users, we've since changed "Change" to "Save".)

```
Storm says, 'what do you do after clicking change?'
Rachael says, 'nothing...it should work..'
```

Rachael here lists out Storm's script and sees this:

```
on hug this
tell context "You hug" + "my name+ + "."
announce_all_but context context's name + "hugs" + my name + my name
   + "."     emote "grins widly and jumps with joy."   end'

Storm says, ' it says I'm missing "end"'
Rachael says, 'hi..sorry...my computer crashed.'[5]
Rachael says, 'you always need to put end at the end.'
```

[5]The version of MacMOOSE Rachael was using was still somewhat unstable. The application did not become reliable until summer 1996. Rachael knows that she may not have heard everything Storm said, since Storm may have continued typing while Rachael was disconnected.

Storm examines Rally, checks 'parents Rally' and 'parents Jasper' and tries to chparent Jasper to #402, generic greeting creature, a program written by an eleven-year-old boy which serves as the parent object for Rachael's pig. She told me during a later interview that she learned this from the tutorial she read earlier. She gets the syntax slightly wrong (forgetting the word "to") and the chparent doesn't work.

```
   Rachael says, 'sorry it crashed again.'
   Rachael says, 'it needs to have end on the last line.'
```

Storm reads 'help chparent'.

```
   Rachael says, 'like this 1:  on hug this 2:  tell context "You hug" +
      "my name+ + "." 3:  announce_all_but context context's name +
      "hugs" + my name + "."     emote "grins widely and jumps with
      joy." end
   Rachael says, 'I mean..'
   Rachael says, 'you need to have end on a line below the others..'
```

This is somewhat confusing advice. Rachael here checks Storm's last commands. The ability to check someone else's last commands was designed for exactly what Rachael is doing: helping someone figure out what they're doing wrong.

Storm now moves the 'end' to a separate line of her program, and also correctly chparents Jasper to generic_greeting_creature.

```
   Rachael says, 'I'm sorry..but I need to go...any questions...?'
   Storm says, ' nope! Thanks a lot, Rachael! Bye, Rally, Clover!'
   Rachael says, 'bye...see you again soon okay! Can you come by
      monday?'
   Rachael says, 'Or this weekend?'
   Storm says, 'probably!'
   Rachael says, 'By the way, this monday most people won't be
      here...sorry. (because of vacation)'
   Rachael says, 'bye!'
   Storm says, 'i'll be there, though!'
   Rachael says, 'okay..see you then!'
```

Rachael now goes home, and Storm tries to hug Jasper. She gets this error:

```
   OOPS!:  Can't find an object named "my name+ + "
        :  Called from Jasper (#913):hug
        :  Line: <2> tell context "You hug" + "my name+ + "."
```

She gets rid of the stray quote before the "my" in line 2, and next gets an error message from the missing return in line 3. She changes it a few times, and finally gets it to compile, but now gets a run-time error. Next she simplifies the script, eliminating the last few lines:

```
on hug this
  tell context "You hug" + my name + "."
  announce_all_but context context's name + "hugs" + my name + "."
```

This gives her a “missing end” error. She now changes it to:

```
on hug this
  tell context "You hug" + my name + "."
  announce_all_but context context's name + "hugs" + my name + "."
  emote "grins widely and jumps with joy."
end
```

This compiles, and she tries it out:

```
--> hug Jasper
You hugJasper.
Jasper grins widely and jumps with joy.
```

She edits the program to add in the missing space after the word “hug.” Next she looks at herself, adds a few adjectives to her description, and changes her gender from royal to female. She looks at Jasper and hugs him one more time, goes home, and then disconnects for the day.

## 4.3 Saturday: Independent Progress

On Saturday, Rachael is on a few times during the day; no one else is around.

Storm logs on in the late afternoon, sees that no one else is on, and wanders around a bit. Next she creates a new creature and calls it Callie. She sets Callie's gender to female and Jasper's to male. She describes Callie as “You see a large, furry cat, with big orange eyes and a royal air around her.” (She uses the client to edit Callie's description property this time, instead of using the “describe” command as she had previously.) She adds this script to Callie:

```
on tickle this
  emote purrs
end
```

That works, so now she adds this one:

```
on utter "cat"
  say "I am not a mere cat! I am deeply offended!"
  emote "looks indignant and raises her nose haughtily."
end

on utter "Callie"
  say "It is a beautiful name, is it not? Just like me."
  emote "purrs and sits down in a regal pose."
end
```

MOOSE's pattern-matching parser makes it easy to have multiple scripts with the same name. These new scripts both work as well. When you type "utter cat," everyone in the room sees:

```
Callie says, "I am not a mere cat! I am deeply offended!"
Callie looks indignant and raises her nose haughtily.
```

She looks back at Jasper's hug script again, and next changes Callie's utter scripts to be:

```
on utter "cat"
  tell context "Callie says, 'I am not a mere cat! I am deeply
   offended!'"
  announce_all_but context context's name
  emote "looks indignant and raises her nose haughtily."
end

on utter "Callie"
  tell context "It is a beautiful name, is it not? Just like me."
  announce_all_but context context's name
  emote "purrs and sits down in a regal pose."
end
```

In her first version, she was able to put the now familiar "say" and "emote" commands—commands she uses all the time to communicate with Rachael—into her program. Her second version makes her program more like the other program she has seen before. However, she doesn't understand what these commands really do. In this version, when you run the "utter Callie" script:

```
--> utter Callie
[The context (the person typing the command) sees:] It is a beautiful
   name, is it not? Just like me.
[Everyone else but the context sees:] Callie
[Everyone sees:] Callie purrs and sits down in a regal pose.
```

Next she changes the "utter cat" script:

```
on utter "cat"
  tell context "Callie says, 'I am not a mere cat! I am deeply
   offended!'"
  announce_all_but context context's name +  "looks indignant and
   raises her nose haughtily."
end
```

Now the output is:

```
--> utter cat
[Context sees:] Callie says, 'I am not a mere cat! I am deeply
   offended!'
[Everyone else sees:] Callielooks indignant and raises her nose
   haughtily.
```

In her first version, the words “Callie says” were generated automatically by the say command. When she switched to “tell” instead of “say,” those were missing. She now has manually added them back in. Storm tests this version and observes that she doesn’t see Callie look indignant any more, so she puts it back to an 'emote' and makes a similar change to the “utter Callie” script:

```
on utter "Callie"
  tell context "Callie says 'It is a beautiful name, is it not? Just
   like me.'"
  emote "She purrs and sits down in a regal, but dainty, pose."
end
```

This doesn't do quite what she wanted:

```
--> utter Callie
[The context sees:] Callie says 'It is a beautiful name, is it not?
   Just like me.'
[Everyone sees:] From Storm:[6] Callie She purrs and sits down in a
   regal, but dainty, pose.
```

Next she changes “tell context” in the second script to “tell Callie”. When she tries it, that line simply doesn't appear to Storm. She puts it back to “context”. In the emote, she changes the “She” to “Callie”, so now she gets a line beginning with “Callie Callie”. Next she changes the start of the line to “emote say.” This prints out “Callie say Callie”. Next she simply scrunches it into one long “tell context” command:

```
on utter "cat"
  tell context "Callie says, 'I am not a mere cat! I am deeply
   offended!' Callie looks indignant and raises her nose haughtily at
   you, with a sniff of disgust."
end

on utter "Callie"
  tell context "She says 'It is a beautiful name, is it not? Just
   like me.' She sits in a regal pose, with an air of such splendor
   that makes you feel unworthy of her presence."
end
```

Now only the person typing the command sees Callie’s response. Storm may not understand that other people won’t see it. In fact, her original version was probably closer to what she wanted. She has spent 29 minutes experimenting.

Satisfied, she wanders around the world a bit. She chparents Callie to be
a greeting creature, as she had done with Jasper. She logs off for the day.

[6] Storm is holding Callie. The system prefaces the output with the words “From Storm” to identify its source.

## 4.4 Sunday: More Mutual Reinforcement

On Sunday, Storm connects again in the morning. Her character wakes up in her home, which she notices she hasn't yet described. She describes it as:

```
You are in a black room, with silver stars all over the ceiling. In
one corner, you see a little pond with a fountain and lots of lily
pads. In another, there is a large, comfy black chair with long,
black cat fur all over
```

She modifies this a few times, settling on:

```
You are in a black room, with silver stars all over the ceiling. In
one corner, you see a little pond with a fountain and lots of lily
pads. In another, there is a large, comfy black chair with long,
black cat fur all over it. There are two open windows, and s light
breeze gently flaps the black lace curtains. Silver bells tinkle
merrily.
```

Next she reads the dog tutorial parts 1 and 2, and makes herself a child of generic dog named Toby. She adds an “on tickle this” script to Toby which just “emote laughs”. She next changes it to this:

```
on tickle this
  say "That is beneath my dignity."
  emote looks at you with scorn and contempt.
end
```

She looks at the pet script on Toby's parent generic dog, and then uses that as a model to improve her tickle script:

```
on tickle this
  tell context "You tickle " + my name + "."
  announce_all_but context context's name + " tickles " + my name +
   "."
  say "That is beneath my dignity."
  emote looks at you with scorn and contempt.
end
```

The previous day, she struggled to use another object’s program as a model. This time, she is successful. This program will tell the context “You tickle Toby” and tell everyone else that “<context’s name> tickles Toby.” The previous day’s experimentation has paid off. It’s worth noting also that her program is not an exact copy of what the tutorial or existing dogs do—she has customized it, making her dog react with a different personality.

She next tries a couple simple string constructions out at the command line, as suggested by one of the system tutorials (the second part of the dog tutorial):

```
--> "You pet " + my name + "."
=> "You pet Storm."

--> my name
 => "Storm"
```

She next reads help on the dig command, and digs a new room called Cat's Corner, and describes it:

```
You are in a smaller black room, and cat pictures are all over the
place!  Toys and a food dish decorate the floor. A well used
scratching post is in one corner. A big, black sofa with red cushions
is on your right. Lots of cat fur is shed all over the place. There
is a window by the sofa, where bird song is drifting in.
```

Unfortunately, she realizes that she accidentally put that description on her original room instead of the new one. With a bit of struggling, she manages to switch the descriptions. She told me during a later interview that she did this by reusing different describe commands from her input buffer to switch them.

Next she digs Jasper's Pond off of Callie's Corner. She is careful this time to make sure she is describing the right room. She describes it as:

```
Water splashes into the large pond in the center of this room!
Several frogscroak amid the weeds and stones. You can see some
minnows swim  racefully in the water. Lily pads are abundant here.
```

Storm wanders around the world a bit and then makes a new object inheriting from generic ticklish object. She told me later that she believes she saw the object number on her visit to the library. She immediately recycles the new object without trying it out. She walks back to the library, looks at the parents of one of the library's objects, goes home, and makes herself another object: catnip mouse, with parent generic thing. She describes it as " A catnip mouse that seems loved well. Made of red felt and a yarn tail."

At this point, Rachael connects and pages her 'HI!!!' They have both been online several hours since they last met, but not at the same time. Rachael audits Storm and examines some of her creations remotely. Storm is in the process of adding a 'throw' script to her catnip mouse. She looks back at the hug script on Jasper as an example. She hasn't responded to Rachael yet, who pages a second time: 'May I join you, oh great power of wind?' Storm pages back 'yes'.

```
Jasper says, 'Welcome Rachael'
Rachael beams in.
Jasper says, 'Welcome Clover'
Clover arrives, following Rachael.
Rachael says, 'hi!'
```

```
Jasper says, 'HiRally'
Rally arrives, following Rachael.
Storm says, 'Hi!'
Rachael says, 'What are you doing?'
StormhugsJasper.
Jasper grins widely and jumps with joy.
RachaelhugsJasper.
Jasper grins widely and jumps with joy.
Rachael smiles.
Storm says, 'Just experimenting'
Rachael says, 'ahh...I see you know about dig.'
Storm says, 'yup'
Rachael says, 'Well, you are learning fast!'
Storm says, 'What do you think of my rooms?'
Rachael says, 'They are very nice.'
Storm says, 'Thanks! Did you see Callie'
Storm says, 'she's my cat'
Rachael says, ' could you tell me what :-) means, or what ever that
   thing like it means? I haven't figured it out, and everyone does
   it!'
Rachael says, 'no I haven't. I'd love too.'
Storm says, 'look at it sideways! It's a smily face!'
Rachael says, 'Oh!'
Rachael grins!
```

On Rachael's arrival, Storm immediately shows off her new creation, Jasper's hug script. Rachael acknowledges it by hugging it herself and then smiling. Throughout the conversation, Rachael offers positive feedback. The exchange is nicely two-way: Rachael has a question that Storm can help her with.

Storm next invites Rachael to her other room, where she shows off her cat Callie. Storm suggests that Rachael say “cat.” Rachael does, and nothing happens. Storm tries to figure out what's wrong. Rachael looks at the code on the cat and figures out that she needs to type “utter cat”, and does. She laughs. This prints output to only her. Storm urges Rachael to try it, not realizing she already has. Rachael explains she did try it—that's why she laughed. They continue to play with Storm's cat, and then Rachael pets her own cat Clover. Storm pets Clover as well and says she's pretty:

```
Storm says, 'Clover is pretty'
Rachael says, 'I think you are worthy of being a
   Crowned_player_class'
Storm says, 'cool! like what?'
Rachael says, 'Well, you'll see.'
Rachael smiles mysteriously.
Storm says, ''wanna see my frog room'
Rachael says, 'sure!'
```

They continue talking, and showing off creations to one another. Rachael explains how her special player class gives you clothes, and suggests Storm use it. Storm explains that her creatures are named after friends of hers—her real life friend Jasper likes frogs. Rachael invites Storm to her game room, which boots you out (sending you back to your home) if you don't type the right thing. She doesn't explain how it works to Storm, but suggests she look at the code to figure it out. After getting booted, Storm just goes idle for a while and doesn't answer Rachael's pages. Rachael goes to work on Clover's enter script, and fixes it. Storm comes back and they talk about the game room a bit. Rachael tells Storm how to use the “list” command to look at code. (Storm looks for help on “list,” which unfortunately doesn’t yet exist. We have added it since then.) Storm says she has to go, and they make plans to meet the next day around 3 PM. They both log off.

Rachael comes back a few hours later, and examines some of Storm's things. She makes a small change to one of her own programs, and logs off. Two hours later, she logs on briefly again. Finding no one on, she disconnects. She connects for half an hour again in the evening, and investigates what some basic system objects do.

## 4.5 Monday: Camaraderie Combats Frustration

Monday (a school holiday), Rachael logs on and off quickly in the morning. Storm connects a little after 1 PM. She starts to work on the dog tutorial, part 3. Rachael notices she's on and page 'Hi!' Storm pages back 'I'm working on a tutorial. When I'm done, I'll page you again. Thanks!' Rachael pages back 'Okay. See you then!!' Storm works through the tutorial, and then recycles the dog she's made. She goes back to working on the throw script on her catnip mouse. She experiments with variations on using say and emote and leaving them out entirely, with little luck.

Rachael works on improving some of her programs, and then snoops into various system functions. She checks Amy's last commands (which relate to the registration room) and then tries to figure out what the registration room does. Finally, she gets curious about what Storm is doing and checks Storm's last commands. Storm has been tossing her mouse over and over. Rachael asks if she can come over, and Storm says 'sure'.

```
Rachael says, 'hi'
Storm says, 'Hi!'
Rachael says, 'Whatca doin'?'
Storm says, 'Being unsucessful with the dratted mouse!'
Rachael says, 'AIIIE! What seems to be the problem?'
[Rachael here checks 'scripts mouse' and tosses it.]
Rachael says, 'Hmm..what do you want it to do?
Storm says, 'well, the mouse should be thrown across the room but it
   just won't GO!!!!'
Rachael says, 'okay this is the script:1:  on throw this   2:
   drop   3:    " thrown across room. "    4:  end'
Rachael says, 'well, first, " thrown across room. " dosen't do
   anything. You need to announce it.'
Storm says, ' what the heck is that supposed to mean?'
Rachael says, 'Pardon?'
Rachael says, 'You need to make it say 'announce " is thrown across
   the room."' '
Rachael says, 'I mean, 'announce my name + " is thrown across the
   room."' '
```

Storm changes the script to this:

```
on toss this
  drop
  announce myname + 'is tossed across room.'
end
```

The old throw script is unchanged, which Rachael points out. Storm tells her to try the toss script.

```
Rachael says, 'You made a tiny error. You said 'announce myname + 'is
   tossed across room.'' when it should say 'announce my name + " is
   tossed across the room."' '
```
[Storm adds in the space.]
```
Rachael says, 'When your done could you please drop your dog? I want
   to test something with Clover.'
```

Rachael tries 'toss catnip' and 'catnip mouse is tossed across the room' is printed to both of them.

```
Rachael says, 'okay, now it announces it.'
Rachael says, 'But you want it to go into the room right?'
Storm says, 'yeah'
Rachael says, 'well, I think that if you said 'this:moveto my owner's
   location' it might work..'
Rachael says, 'instead of drop...let's try it.'
Storm says, 'well, let's lay off the mouse before i strangle
   it!!!!!!!!!'
```

Storm tries to take Rachael's advice, but adds the new code to the first line instead of putting it on its own line. Now you can't toss it at all. In my face-to-face discussion with Storm and Rachael together, we were talking about the benefits of working with someone else, and Storm noted particularly appreciating Rachael's support at this moment:

> It was sort of handy when she came in. Like that catnip mouse, which I finally threw away, cause I was just sick of it. And it wasn't working, so.... It was nice to have somebody who I could just sort of rant and rave all my woes about this mouse to. [Laughing.] We just started saying “it's ok, it's ok, calm down, it's just a little object, it won't kill you.”

Storm drops her smart dog as Rachael requested, and it reacts to Rachael's cat as Rachael had hoped. Storm compliments Rachael on her cat. Storm suggests they go to the game room, and Rachael agrees. This time Rachael explains how it works right away. After jumping around in the game room for a while, Storm suggests they explore. They go to Paradise Island together, a room built by an eleven-year-old girl where you can swim, climb trees, and build a summer home. Storm builds another home there. Rachael teaches Storm about the 'audit' command. Rachael suggests Storm connect her houses in different places together to make a loop. Storm asks how. Rachael reads "help dig" and gives Storm a concrete example of a command. Storm works on making the circle of exits, with help (and positive reinforcement) from Rachael. Storm runs out of quota, and Rachael advises her to recycle stuff or ask Amy for more quota. (Each member of MOOSE Crossing is given a quota for the total number of objects he or she can build. This is necessary to prevent the database from getting too big for the machine. Children who ask for more quota are almost always granted more.)

Rachael has to leave. They plan to meet the next day at 10:30 AM. Rachael logs off and Storm logs off. Rachael logs on again a few minutes later. I am back from Maine, and talk to her a while later:

```
Rachael laughs. "Have you seen any of Storm's things
Amy says, 'she's made stuff already?'
Amy says, 'wow!'
Rachael says, 'Yes! Will almost full credit!'
Amy says, 'she's been busy!'
Rachael says, 'Yep! I've showen her somethings, but she's done some
   on her own.'
Rachael says, 'It is. She is really nice.'
Rachael says, 'We spent almost 2 hours together today.'
Amy says, 'great!'
```

Rachael is logged on for several more hours that day. She programs Marj, a robot that acts as a second self for her online.

## 4.6 Tuesday: Collaboration

Storm logs on early in the morning. She continues to work on the dog tutorial, part 3. As part of her work on the tutorial, she has made a pet dog named Tao and another pet called Melvin the Moose. She changes the parent of her objects from generic dog to generic cliché-spouting object, a program I wrote that inserts clichés into the conversation at random intervals, and to which you can teach new clichés. She disconnects.

Rachael and Storm both connect promptly at 10:30, the time at which they had agreed to meet. Rachael sends her new puppet, Marj, to talk to Storm. Storm teaches Melvin a cliché. Rachael drops Clover. She's made some coding improvements on Clover since they last talked. Each of them is immediately showing off the new work they've done.

Storm must be a bit confused, because she pages Rachael asking if she would like to join her. Rachael has Marj explain that she is already there. As Rachael is talking, Melvin the Moose is generating errors.

```
Marj looks distastefully at Melvin. It is giving Rachael lots of
   errors because she is talking through a puppet.
Marj says, 'I am not a MEAR puppet. I am a non - orgainic lifeform.'
Storm says, 'I am sorry about my moose. She is very...strange.'
```
[What Storm is saying isn't actually getting across to Rachael, because Rachael’s puppet isn’t working correctly yet.]
```
Marj says, 'We non - orgainic lifeforms have feelings too!'
(from Rachael's room) Rachael grins broadly.
Storm says, 'And I know, o mighty one, that you are not a MERE
   puppet!'
```
[Rachael again doesn't hear this line.]

The conversation continues, with Rachael not hearing Storm's side. Rachael wonders why Storm is being so quiet. Storm responds that she's not; Rachael of course doesn't hear that either. Storm asks a question, and gets no answer. She repeats it with increased urgency (i.e. six question marks). Finally, Rachael checks Storm's last commands and realizes something is wrong. In this instance, Storm has helped Rachael to detect a bug in Rachael’s program. The exchange of technical help has already become very much two way.

Storm suggests they make a medieval room. Rachael likes the idea. Storm asks where they should make it. Rachael suggests off of her castle. Rachael makes the room. Storm asks if she can be the one to describe it, and Rachael says yes. Joint ownership of objects is not currently supported. It would be a desirable feature. In this case, Storm ends up building an additional room so that she can be the one to describe it. Rachael sets up the exits to connect the new room to her castle. Storm writes a first description, and Rachael offers suggestions for additions. Storm starts with:

```
You are in a red room. Armor and wepons are all over, but they are
old and dull
```

She expands it to:

```
You are in a red room. Armor and weapons are all over, but they are
old and dull, so you can't use them. But the armor and weapons are
strikingly well made.
```

This conversation ensues:

```
Storm says, 'hi. Look here and tell me what you think...'
```
[Rachael isn't in the room at this moment, and doesn't hear this line.]

[Rachael arrives.]
```
Rachael says, 'hi'
Rachael says, 'hmm...'
Rachael says, 'Great!'
Rachael says, 'The only problem is the exits...look it says
   passage...armory'
Rachael says, 'We can have amy help us fix that.'
```

```
Storm says, 'any other ideas for describing?'
Rachael says, 'wooden benches?'
Rachael says, 'blood stains?'
Rachael says, 'rusty weapons?'
Storm says, 'i like it! give me a second...'
Rachael says, 'a fire'
Rachael says, 'ok'
Rachael says, 'it is dim'
Rachael says, 'the only light is from the window and fire'
Rachael says, 'the fire is roaring'
Rachael says, 'in the center of the room.
Rachael says, 'We could rename the room to "hero's hall!"'
Rachael says, 'I mean Hero's Hall'
Storm says, 'try the description now...'
```

Rachael’s brainstorming inspires Storm to improve the description:

```
You are in a red room. Armor and weapons are all over, but they are
old and dull, so you can't use them. But the armor and weapons are
strikingly well made. Tapastries of unicorns and knights decorate the
walls. There is a lone window here, with red velvet curtains. You see
a blood stain, old and brown, streaked across the wall. The only
light comes from the fire, for in this room it is eternally night.
The fire is in the center of the room, and it is crackling wickedly.
You see a white flutter in a shadowed corner, and your heart stands
still. But it is only a piece of cloth...or so you think...

Rachael says, 'wow! It's great!'
Storm says, 'Thanks.'
Storm says, 'I love writing, you see...'
Rachael says, 'It's like a murder mysteroy'
```

We discussed the construction of the room in our face-to-face interview:

> Amy: Which parts of that were whose idea? How did it start, and whose idea was which part?
>
> Rachael: I think it was her idea originally to build the room
>
> Storm: Yeah, I wanted to build a room that we could both do stuff in. And she suggested a castle. And castles and dungeons and dragons are just stuff I've always liked. When I was like 2 I had a dream about dragons that would take me places. So I always liked them and I thought, that would be pretty cool if we did all this stuff. I wanted to describe it cause I like to write. My original description was like one sentence, which was boring.
>
> Amy: And then what happened?
>
> Storm: Then she [turning to Rachael] then you said all the suggestions, you know, like blood stains and stuff like that. And I thought "Ah yeah!" And I thought of a catalog that we get, it's called Design

> Toscano. It's got gargoyles and tapestries, swords and that kind of stuff. So I just added a whole bunch of stuff.
>
> Rachael: And I think after I saw her description I thought we should add some objects or something in the room or something to follow up the description.
>
> Storm: Like blood stains. And I thought "Ah ha! Yes, There we go! Now I have some ideas"
>
> Amy: Do you think either of you could have done as good a job on your own?
>
> Storm: No.
>
> Rachael: No, I don't think so.
>
> Amy: Is it fun working with someone else?
>
> Rachael: It's much more fun than working by yourself. Cause if there's nobody to see it, why even bother doing it?

This is a nice example of collaboration. The girls concur that the result is better than if either girl had made it on her own. They are learning from each other and inspiring each other to do better work. The result is a positive experience with creative writing: constructive feedback, a feeling of success, increased confidence, and increased interest in pursuing creative writing further.

Rachael and Storm decide the room needs a dead body and note. Is it a murder or a suicide? Who is dead and why? Their shared vision of what has happened here evolves as they work together. This time Rachael writes, and Storm offers detailed feedback. Rachael says she has to go. They agree to meet Thursday at 5.

Rachael comes back half an hour later. She asks Storm if she's going to go the Media Lab. Storm doesn't understand; Rachael explains that some times people go to the Media Lab to do MOOSE Crossing. Storm says no, she's not going. Storm shows Rachael her new Gargoyle. It has a couple of errors: it's an adaptation of the smart dog tutorial, but it's missing some required properties. Rachael explains that to Storm, and Storm fixes it.

Storm has made a ghost object. Its description reads:

```
The ghost of the murdered girl. She says "I will not harm you. I am
left here to mourn my loved one forever. Heaven would not take me
```

```
because of my suicide, but Hell could not take me because of my
goodness. So I must stay here, and mourn Patrick until the stars grow
cold and the Earth is no more."
```

Rachael suggests Storm might make the ghost say that whenever anyone enters the room. She explains how to write an 'enter' script.

The previous day, I met and talked with Storm online. She showed me her creations, and I complimented her on them and on her fast progress in general. Storm mentions this encounter to Rachael. Despite the child-centered nature of the environment, feedback from adults is still significant:

```
Storm says, 'oh, amy loved my work and gave me some more quota to
   make stuff..'
Rachael says, 'That's great!'
Rachael says, 'she did the same to me, byron, zoro, and miranda.'
Storm says, 'I'm going to try to make that script... Why don't you
   create a sword or something? Make it out of Generic Jewelry, if
   there is such a thing...'
```

Rachael and Storm hang out together in the same room. Rachael works on getting Marj to be able to page people; Storm writes the enter script Rachael suggested. The girls' online friendship continues to the time of this writing, much in the same fashion.

## 4.7 Seven Months Later: Meeting Face to Face

The girls finally met face to face for the first time on the day that they both came to the Media Lab to talk with me about this chapter. They hadn't met previously because they live more than a 90-minute drive apart, and because the MOOSE Crossing code of conduct strongly discourages meeting other members face to face for safety reasons. Storm said Rachael was much like she had imagined. Rachael confessed that she hadn't really given much thought to what Storm was like in real life. Storm arrived at the Media Lab first, and I interviewed her separately. I had interviewed Rachael on several previous occasions. Next Rachael arrived, and I interviewed the two of them together. I then gave them both an hour to talk without me present, and went to chat with their parents. The girls immediately decided to log on to MOOSE Crossing. Afterwards, I asked them to compare connecting together versus connecting alone from home:

> Amy: How is doing this sitting next to each other at two terminals different from doing it from two towns a hundred miles away?
>
> Storm: It can be a lot more funny, cause we can talk to each other.
>
> Rachael: Yeah, like when they're at the Media Lab, at least in the beginning especially, they'd be talking and they'd say like “that's not

> funny." And I'm like, what are they talking about? And then when I went to the Media Lab,[7] I realized they're joking around at the same time. So we were just doing that to Zeus. I told him eventually what was going on.
>
> Amy: So it's more fun with someone in the same room?
>
> Storm: It's a lot of fun that way.
>
> Amy: Do you think it would be just as much fun if everyone was always in the same room?
>
> Rachael: Well, no.
>
> Storm: That won't work if everyone was just in the same room.
>
> Amy: Well, it *could* be. We could just have the people at the Media Lab and that's it.
>
> Rachael: It's fun to have people you've never seen before. And it's fun, the aspect of it being global.

Rachael and Storm hit it off face to face. After leaving the Media Lab late that Sunday afternoon, their families went out to dinner together. A few days later, I asked them each via MOOmail (email internal to MOOSE Crossing) how she thought meeting face to face would affect their online friendship. Rachael responded "I'm not sure. I think that it might strengthen our friendship...but I honestly don't know." Storm said "I don't think it really changed our friendship much at all." They hope to meet up face to face again, but the driving distance between them may be an obstacle.

Rachael's experience meeting Storm stands in contrast to her experience meeting Miranda. Rachael describes herself as "unpopular" and says "I'm a very intellectual person. I'm not very physical." I would characterize Storm as earthy. On meeting face to face, they appeared immediately comfortable with one another. On the other hand, I would characterize Miranda as a popular child. Miranda is smart, pretty, and adept and relaxed in her social relations. Rachael and Miranda had developed a warm friendship online. However, on meeting face to face, they seemed immediately uncomfortable with one another. Online, they hadn't been aware that one of them was a popular sort of kid, and the other was unpopular. Rachael says that after meeting Miranda face to face, she was initially much less comfortable with her; however, over the months that followed, she slowly became more

[7]Rachael is referring to the children who come to the regular MOOSE Crossing after school program at the Media Lab. She attended that program occasionally—three times in 1996.

comfortable again. Different factors are significant in determining social success and social compatibility online versus face to face.

## 4.8 Conclusion: Integrating Technological and Social Contexts

Storm’s experiences that weekend are quite typical of the general patterns of interaction on MOOSE Crossing. However, she was unusually lucky in one key respect: she immediately met someone with whom she has significant shared interests. The population of MOOSE Crossing is still smaller than is desirable (primarily because not enough kids have Macintosh computers on the Internet, and also as a result of the cumbersome nature of the registration process). This makes it more difficult for individuals to meet people they particularly like, because there are fewer people to chose from. Even within a larger community, it may take time before a new member happens to meet someone with shared interests. Storm was fortunate to meet Rachael right away.

Over that weekend, Storm used a variety of learning strategies and sources of support. In this transcript she makes use of:

- the introductory message sent to all new members with their passwords,
- the online help system,
- online tutorials,
- unsolicited support from peers,
- solicited support from peers,
- trial and error, and
- using others’ projects as models.

Of particular interest is the inseparability of the social and intellectual activity going on. It matters not just that a learner have access to adequate support. One must also ask, support from whom or what? What is the relationship between the learner and the person or program offering assistance? That relationship is rarely if ever neutral. If a computer program attempts to affect a disembodied, non-anthropomorphized tone, working with that program will still evoke the learner’s basic feelings about computers and technology in general. For many people, those feelings are not positive. For this reason, the tutorials on MOOSE Crossing are written in a first-person, chatty style. The dog tutorial begins, “Hi there! This is Amy and I thought I'd show you how I made my dog, Pumpernickel.” (Storm noticed the tone of the tutorials, and commented that it was just right—”It's not totally perky, but it's not like [speaking in a monotone] ‘this is how you do it.’”) The informal, warm tone is intended to put kids at ease.

While some attention to the “personality” of a computer program can help make a more supportive learning environment, there is no substitute for human contact. A human can, for example, tailor assistance to the particular situation. Rachael was responding to Storm in a way computer programs can not, at least yet. Newman *et al* (Newman, Griffin et al. 1989).comment that "current computer systems are actually quite far from being able to perform the feats of sensitive interpretation performed routinely by human teachers"(Newman, Griffin et al. 1989). Research in artificial intelligence and education aims to help tailor support to the user, often by attempting to model what the user knows and anticipating typical mistakes. Unfortunately, such systems are still largely unsuccessful, and it's unclear how much progress in this area is even possible. Unless the general project of creating an artificial intelligence is successful, interacting with intelligent tutoring systems will remain unsatisfying. A teacher's relationship to a student is fundamentally social in nature, and computers can not yet fill that role. It's not likely they will ever be able to do so.

Storm received help not from just anyone, but from a girl her own age who also loves *Star Trek* and *Monty Python*. Similarly, she used as examples not some official library of sample programs written by a disembodied authority figure, but programs designed and actively used by her new friend, Rachael. She also looked at additional projects that Rachael recommended, written by Rachael's other online friends. Her choice of examples to embrace is a significant act that takes place in the context of a network of social relations. A child who mimics another's way of dressing or speaking is paying that child a high compliment, and taking on that child as a role model. Similarly, using someone's project as an example to learn from is to some extent accepting the project's author as a role model. Making use of a sample program can be as much a social as an intellectual act.

Social context is of central importance to any learning experience. One of the strengths of networked learning environments is their ability to help integrate a supportive social context with the computational context. The Logo language traditionally arrives in a shrink-wrapped box, with little social support for its use. Teacher workshops try to build networks of support for the use of Logo as a learning tool; however, the workshops are usually too short, and too few teachers can attend. Once a workshop is over, typically so is the support. Additionally, the model most often used is one to many: a workshop leader supports a group of teachers; each teacher supports many kids. In a many-to-many model, kids support one another. MOOSE Crossing attempts to deliver a rich computational artifact to kids, with a network of many-to-many social support built in.

# 5. Community Support for Construction

*"Another thing about moose-crossing is that I feel as if I can really help someone. I like learning and doing stuff on my own, but the real reason I come to moose-crossing is that I feel needed, and wanted. While programing is a lot of fun, I don't think I'd do it, if there wasn't anyone who would appreciate it."*
*— Rachael (girl, age 13)*

## 5.1 Uzi's Frustration

Uzi (boy, age 13) was excited about joining MOOSE Crossing, and his mother was delighted. She wrote, "He is 13 and is doing a lot of MUD'ing lately. I'd be thrilled to have that energy turned towards something more 'scientific.'" Approximately six percent of MOOSE Crossing members were already members of other MUDs before joining. Before he even sent in his permission forms, he was mailing me detailed questions from his mother's account:

```
Thank you for the help!  By the way, this is Uzi speaking.  How do I
program socials to do different tthings with different subjects?  For
example, I type safe jed".  I see " You drop a fifty ton safe on
jed", Jed sees "  AAAAAAH! there is a safe falling on you!",  and
everyone else in the room sees "SQUISH!".
```

He mailed in his permission forms, and first connected a few days later. No other kids were online at the time. A veteran MUDder, he knew immediately he wanted to describe himself and set his gender. Unfortunately, a couple days earlier I had changed the syntax of the "gender" command. (It was changed from "gender me is <gender>" to "gender me as <gender>" to avoid confusion with use of "is" as an infix operator.) I announced the change in the newspaper, but neglected to update the help message for "gender." Uzi tried several variations on the command to no avail. He finally pounded out on his keyboard "kill self." This, amusingly enough, printed out some help on how correctly to kill a task (running program). Uzi was used to MUDs where killing people and things was normal; on MOOSE Crossing the command translates to an obscure and rarely-used advanced technical command.

This was relatively early in the development of MOOSE Crossing, and the population was still small. Most kids knew to come by on Monday afternoons, when there were always lots of other kids around. On this Friday afternoon, only Case, an MIT undergraduate working on the development of the MacMOOSE client program, was logged on. Uzi couldn't figure out how to contact him. Uzi wandered around the virtual world a bit, and then gave up. Three days later he connected again at 9:30 PM. This time, no one at all was logged on—most MOOSE participants are younger than Uzi, and 9:30 PM is quite late. He wandered around the virtual world for forty minutes, and then gave up for good.

Uzi's experience was fraught with problems. His past MUDding experience worked against him—instead of consulting the list of basic MOOSE commands sent to him with his password, he kept trying commands from other MUDs that did not work. That gap was cultural as well as technical—he spent a significant chunk of his time repeatedly trying to kill things. Most kids intuitively understand from the code of conduct that killing is not allowed. Uzi was so accustomed to killing things on MUDs that he failed to make that inference. Most importantly, Uzi failed to meet anyone else online.

MUDding alone is rather like bowling alone (Putnam 1995)—it misses the point of the activity. Other children could have helped Uzi with his technical difficulties, and shown him around the virtual world. He did wander by a number of significant projects made by other children, but paid little attention to them. Those projects would have been much more interesting if proudly presented by their owners. His experience is in singular contrast to that of Storm, who was practically overwhelmed by Rachael's eagerness to help her. In this chapter, I detail ways in which community supports construction activities.

## 5.2 Pilot Study: Programming for Fun

In 1993-1994, I conducted an informal study of twelve adults who learned to program for the first time on MUDs, just for fun. Eight learned on MediaMOO. The twelve subjects programmed a variety of objects. Jim is a 43-year-old building contractor who lives in the Pacific Northwest of the United States. He built a "77 Jeep Cherokee." Susan is a thirty-eight year old professor of agriculture from the South. She built a simulation to help her students learn to perform certain laboratory procedures. Christopher is a 23-year-old graduate student in comparative literature in South Africa. He programmed a poetry generator. Neil is a 30-year-old graduate student in American cultural studies in the Midwest. He built a "RetroFuturist Aerodrome" complete with a biplane, a blimp, and a taxi stand for those who want to catch a ride.[1] I'll describe Jim's experiences in detail.

Jim has always lived in the Pacific Northwest of the United States. After he graduated from high school, he was drafted into the army. By a stroke of luck, he narrowly avoided being sent to Vietnam. After two years of military service, he attended two years of junior college on the GI Bill. He took what

[1]Parts of this section appeared in (Bruckman 1994).

he describes as “just the fun classes” and never got a degree. In need of work, he learned that a friend of a friend needed help building a house. After a number of years, he built up enough expertise to start his own building contracting business. Jim has little mathematical or scientific background, and had never programmed before he encountered MediaMOO:

> I work on a computer at home. I don’t have it in an office, at school, or anything like that. We use a Macintosh. My wife was a graphics designer and she needed a computer to stay in her field. I’m a building contractor. She bought a Mac about four years ago and started doing design on the computer, and I started using it for my business, just playing with it, and really liked using it.

Asked why he decided to try programming, Jim describes a desire to contribute to the community:

> Since I started getting involved with this I became interested in programming because I wanted to be able to manipulate this world in a better way. And instead of just looking at things and creating things that already existed, I wanted to try and put my hand to actually creating something of my own.

Since the MOO language is object-oriented, it is easy to make useful objects simply by making something which inherits from an existing parent object. By “creating something that already existed,” Jim means making an object which inherits its functionality from someone else’s object. You can make a television set with one simple command, making something which has generic television set as its parent. Making something which inherits from an existing parent object is an easy first step towards programming. It is part of what encourages people that they can and would like to make something of their own.

The virtual world on MediaMOO is filled with interesting objects made by members. These serve as models for new users. Every object in the system functions as a possible model to learn from and be inspired by, just like every page on the World Wide Web is a potential model for learning HTML. It’s significant that each model is closely associated with its author, whom you might happen to meet online. Examples to learn from are situated in a social context.

Jim also describes the MOO environment as an interactive story to which he wants to be able to contribute:

> It’s like reading a story that I get to help write. And I wanted to get more input into what was happening, beyond just chatting with people

> and creating a box or whatever... I wanted to try and do something with it.

The community of learners present on MediaMOO was one of his key motivations for learning to program. He wanted to be able to contribute something to that community. Objects are conversation pieces in MUDs. When Neil flies by in his blimp, others stop to admire it and ask him how he created it. That becomes the basis for striking up a conversation, rather like dog owners in a park being able to strike up a conversation about dogs and then segue to other topics. Making an interesting object confers status within the community to the object's designer. Jim wanted to build something of his own rather than simply admiring everyone else's work. Analyses of individuals interacting with standalone computer systems such as those in Sherry Turkle's *The Second Self* often focus on the individual's quest for mastery (Turkle 1984). That quest takes on added dimensions when the activity is strongly situated in a supportive community context. Other residents of the virtual world form an appreciative audience for work. In addition to wanting the personal satisfaction of having mastered the technical environment, Jim also wanted to be able to use his new creation as a catalyst for social interaction and conveyer of added social status. On a more altruistic level, he said he wanted to contribute something to a society from which he has benefited. People's motivations for trying to program something in MUDs are primarily social.

The first step in learning to program is perhaps the hardest. The initial barrier is primarily emotional. Most adults who do not have formal technical education suffer from some degree of technophobia. On MediaMOO, a user named "cdr"[2] made a set of clear, simple tutorials. Seven of the eight MediaMOO programmers interviewed began by doing one of these tutorials. Susan comments that for her, the tutorials' primary function was to help her overcome that initial emotional barrier: "I did cdr's tutorials and I realized 'I can do this!' But then I had to step back and figure out what it was that I had done."

Cdr began by programming a "television set" object. You can turn on your television set and receive different "programs" on different channels. One of the first shows he added to the television schedule was called "Coding with cdr," an instructional video teaching you how to program a box of donuts that you can eat. The metaphor of an instructional video seems to put people at ease. The success of these tutorials also depends on cdr's friendly, warm personality. His manner puts people at ease. The tutorials are not detached technical instructions, but help from a caring, supportive person. Jim enjoyed doing the tutorials:

[2]The character "cdr" is Dr. Kenneth G. Schweller, Professor of Computer Science and Psychology at Buena Vista College in Storm Lake, Iowa.

> I'm totally program-language illiterate. Well, I found cdr's tapes. Cdr was able to lay out exactly what was going on and lay it out step by step—you watch his tape, and you do what he says. And it worked! I was able to start to understand what the basics of it was all about. I made his box of donuts, and then I made myself a car to drive around in. And I added some events so I personalized it somewhat. I really enjoyed that.

Jim was excited enough about his car to want to port it over to another MOO where he has an account, PostModern Culture MOO (PMCMOO). He discovered that one of the needed *parent objects* for the car, generic portable room, did not exist on PMCMOO. In solving this technical problem, he relied on the help of other members of the community:

> I got excited about programming just by making cdr's car and doing those things, to where I went back to the other MOO where I had an account and made myself a vehicle there from scratch. Cause they didn't have portable rooms there. I had help, of course. I got a couple people who knew more than me to give me a hand when I got stuck. Being in the MOO enables you to be... two people can work on the same project at once. I could work on my verbs, and when I had a problem I would call somebody over, and they could just list it out and they could say "gee, you need to do this" or "you need to do that." It was just like having someone sitting behind me at my terminal critiquing what I was doing.

Just as the community provided Jim's initial motivation for learning to program, it also provided support to help him through the process. In a MUD, you are almost never alone. If you have a technical problem, there are other people there to help you. Helping others is a central activity in MUD culture; it is a basis on which strong friendships are formed. People often speak warmly of those who helped them through technical difficulties.

Jim was inspired enough by his programming experiences to want to share them with teachers at his children's school:

> I plan on bringing high school kids online. I'll be working with a high school this fall. We're just getting the Internet connection. What I'm trying to do is get the teachers interested. I mean they're all interested in Internet because of all the research and all the facts that can be gleaned from there. But I also want to introduce them to the MOO just because it gets people excited about programming.

"This has inspired me," says Jim. He explains that he has just enrolled in a programming course at his local university's extension school.

For Jim, the online community provides:

- Role models,
- Situated, ubiquitous project models,
- Emotional support to overcome technophobia,
- Technical support, and
- An appreciative audience for his work

Each of these factors contributes to the individual's motivation to learn and ability to learn. In the rest of this chapter, I'll discuss each of these factors, and then conclude with a detailed analysis of an extended interaction among three children.

## 5.3 "Did You Really Make This?": The Importance of Role Models

Jim didn't immediately see himself as the sort of person who could learn to program a computer. He is, after all, "just a building contractor." Many assume that programming a computer is a difficult activity that should be undertaken only by the technically-educated elite; it's not the province of mere building contractors—or humanities majors, or women. Technology increasingly surrounds our everyday lives, but most people can't imagine themselves having meaningful control over it.

For girls and women, the problem is compounded: they may fear success with computers as much as they fear failure. Sherry Turkle writes that "the computer becomes a personal and cultural symbol of what a woman is not" (Turkle 1986). People define their sense of self in part through their activities. The cultural connotations of "computer programmer" do not conjure up Jane Fonda or even Jane Pauley. Contrast the connotations of programmer and figure skater. By becoming a successful figure skater, a woman comes closer to an American cultural ideal of womanhood. By becoming a successful computer programmer, a woman drifts farther from that ideal.

Positive role models can help to counter this problem, for men as well as women. Jim found himself surrounded by peers who could program. He decided to give it a try himself not only because he knew those friends could help him, but because he could imagine himself being like them.

The first time Miranda (girl, age 10) tried MOOSE Crossing she stopped and asked somewhat solemnly, "May I ask you something?" "Yes," I replied. She asked, "Did you really make this?" Her face glowed when I replied that I had. In the weeks to follow, her mother tried to take her to a concert at MIT. Miranda asked if I would be there. Her mother said she didn't think so. Miranda replied that then she didn't want to go! I had clearly become a role model for Miranda. Another eleven-year-old girl recently concluded a visit to the Media Lab with the exact same question. Both girls probably already knew or strongly suspected that I had designed and programmed MOOSE Crossing, but the explicit confirmation was satisfying. If you have any doubts about whether women can program computers, or whether it's a proper thing for a young woman to do, look around: there are women and girls all around the virtual world of MOOSE Crossing. The person who designed the system is a woman. These strong female role models encourage girls to become involved.

It's worth noting that MOOSE Crossing is a multi-age community. Role models can be of any age, but are most commonly older. Younger children are motivated to learn by their desire to be like the older children, and learn by imitating them. Aaron Falbel eloquently describes an older student teaching a younger one to read in his study of the Friskolen 70 school in Denmark:

> Maria (11) and Johanne (9) are using the wood-burning set in the workshop to make Christmas presents for their friends and relatives. Maria has just completed a small, wooden tic-tac-toe game for a cousin of hers in Sweden. The X's and O's are made out of wooden pieces that fit neatly into a finely crafted playing board. She is etching an inscription to the recipient of the gift on the back of the playing board when little Clara (6), who just started at the school at the beginning of the month, wanders by and is drawn into the scene. She watches Maria etch a floral design under her inscription and is fascinated by the strangeness of this smoldering writing instrument. Maria notices the presence of Clara and displays for her the finished product. Clara turns over the board and feels the grooves of the inscription with her fingers. Maria asks her, "Can you read?" Clara shyly shakes her head no. Maria smiles and says "Come," and she motions for Clara to come sit on her lap. Then, very slowly and sweetly, Maria sounds out the words as Clara guides her fingers over the dark-brown letters. Clara is totally absorbed: her face conveys an expression of rapt concentration, her mouth partly open, and her cheek leaning against Maria's arm. The entire episode lasts not much more than a minute. It is so effortless, natural, and unself-conscious that to call it "peer tutoring" would be to debase the beauty of the situation. (Falbel 1989)

At Friskolen 70, students of all ages learn together in one open classroom. The age segregation typical in the school systems of most cultures is often counterproductive.

On MOOSE Crossing, two thirteen-year-olds, Jack and Rachael, have become particularly strong role models for younger children. They are both accomplished programmers, and their imaginative creations are often catalysts for social activities. In particular, Rachael has founded a sub-community she calls "The Gathering" for medieval role playing. Other children are often heard to comment that they wish they could type as fast as Rachael, so they could keep up in the conversation. It's evident that many of them want to be like Rachael in a variety of ways. This benefits the younger children, and also supplies Rachael with some strong positive reinforcement. The community provides members with a ready supply of positive role models.

## 5.4 Situated, Ubiquitous Project Models

Role models can be a key source of motivation for learning. Jim wanted to be one of the people who has made something, like Neil and cdr. He wanted to be like them, which inspired him to want to take on a new intellectual challenge—and later to bring that same challenge to students at his children's school. Clara wanted to be like Maria, so she developed an interest in learning to read. Jim and Clara both also took interest in the intriguing artifacts they saw around them. The novelty of cdr's "tapes" and Maria's wood carving contributed to their motivation to learn. It's important to note that these examples were not simply unattributed entries in a library. It wasn't just any old wood carving; it was *Maria's* wood carving. The relationship between Clara and Maria was part of what made Maria's wood carving a compelling model for Clara to learn from. On MOOSE Crossing, Mouse (girl, age 8-9) wanted to create Charlie the wiggly caterpillar because she wanted to make something just like her older sister Miranda's wiggly snake. The object was interesting in relationship to her admiration for its owner. Role models and project models are tightly linked. People establish their identity within the community (and their desirability as a role model) in part through the objects they've created. Conversely, objects are considered intriguing in part by virtue of who created them. Examples are situated in a social context.

It's important that these sample projects are not isolated off in a library—they are ubiquitous. Most programming languages and scripting tools come with a separate folder or directory of examples. To see the examples, you must deliberately seek them out. On MOOSE Crossing, every object in the system can function as an example. In this respect, MOOSE is rather like the World Wide Web. You can usually view the HTML source for any document. When you want to know how to do something yourself, you can remember something you saw which uses that technique, and look at the HTML source code. The benefit comes not from the large number of examples (though this doesn't hurt), but from the fact that as you are going about your daily business using the Web or interacting on MOOSE Crossing, you are developing an ever richer vocabulary of models you can refer back to.

The community of MOOSE Crossing is of course multiple orders of magnitude smaller than that of the World Wide Web. It's akin to comparing the impact of new architectural forms in a small town and a big city. An innovative creation in a small town will be noticed by a high percentage of the population, and is likely to have a strong influence on the town in the future. An innovative creation in a city might be seen by more people total, but by a lower percentage of the population, and is likely to have a less discernible impact on future designs in the area.

Since the set of examples to learn from is the set of all things everyone in the community has ever made, those examples reflect the interests of the community. That set of examples is not static; it grows and evolves as the community grows. A single, centralized author of a set of examples could never reflect the community’s interests and needs as well as the decentralized process of using everyone’s creations as examples to learn from.

A particularly good example of sample projects that are ubiquitous and strongly associated with their creators are people’s pets. On MOOSE Crossing, it’s possible to make an object follow its owner around through the virtual world. This seemingly minor feature has proven to be unexpectedly important. Recall the conversation between Rachael and Storm referred to in Chapter 4. Rachael (the experienced user) is showing Storm (the new user) around the virtual world. As they are exploring, Rachael is being followed by her pet pig Rally and her pet cat Clover. This conversation ensues:

```
Rachael has arrived.
Rally says, 'Hello Clover'
Rally arrives, following Rachael.
Clover arrives, following Rachael.
Rachael says, 'I suggest n'
```

[Rachael is suggesting they continue their explorations by going north. “n” is usually allowed as an abbreviation for north.]

```
Storm says, 'how do you make animals?'
Rachael says, 'Well, it depends if you want a new type of animal or
   one that already exists.'
Storm says, 'new type'
Storm says, 'i'd like an animal to follow me around'
```

Experienced users are almost all being followed by a pet (or several pets). New members immediately encounter these pets, and usually decide they want one too. Pets that follow their owners are a compelling example for this particular community. The more broadly applicable design principle is the value of sample projects that are constantly visible, immediately associated with their creators. This relatively subtle design decision has catalyzed the process of engaging new users in making things. Compelling sample projects made by peers enhance motivation for learning to make things. Making those sample projects more visible (people’s pets are everywhere all the time) creates additional opportunities for this positive effect. A pet that follows you around is also very prominently associated with you, which leverages on the benefits of linking role models and project models.

## 5.5 Emotional and Technical Support

One school of research in technology and education aims to apply artificial intelligence techniques to give the learner support. Typically the system makes a model of what the learner understands, and tries to anticipate and correct for common misconceptions. Existing systems have had limited success in this modeling process. However, even if the modeling were much more successful, they would still suffer from a deeper flaw: they fail to account for the fundamentally social nature of learning. Asking for help, receiving help, and giving help are all social acts which help to build networks of relationships. Help is not merely information. Getting help from the computer is not the same as getting it from a person. The relationship between the tutor and the tutee is an essential component of the process.

Peer tutoring provides a great deal of positive reinforcement for both the tutor and the tutee. After Rachael (girl, age 13) helps Cindy (girl, age 11) to finish her animal shed and posts an advertisement about it for her, they have this conversation:

```
Cindy says, 'Thanks so much...what can I do to hellp you?
Cindy says, 'You help me too much...'
Rachael says, 'well...let me send it, and then we can talk!'
Cindy says, 'K. Thanks again'
Rachael says, 'oh, no I don't. It's all in the line of duty!'
Rachael says, 'I was and am helped...now I help others!'
Cindy grins
Cindy says, 'You certainly do'
```

It's true that Cindy needed technical assistance and received it, but this unusually self-reflective snippet of conversation reveals some of the richness of the social context in which giving and receiving help takes place. Exchanging help is part of the social bond between Cindy and Rachael. Cindy knows not only that she can get the answer if she has difficulty, but that she can get the answer from a caring friend who enjoys helping her. The emotional support Rachael provides is inseparable from the technical support. Rachael has received positive reinforcement for both her generosity and her technical skill. When explicitly asked what she wants in return, she suggests that they just spend some time together. Cindy is more than happy to do so. The girls talk for 40 minutes afterwards, until it's time to go have dinner. They talk about their ideas for MOOSE Crossing projects, about school, about their friends. They exchange secret nicknames that only their closest friends use. Receiving help from someone you would tell your secret nickname is clearly very different from receiving help from a computer program or a schoolteacher.

Most of the time in most schools, adults teach and children learn. Learning from peers is refreshing because it equalizes the power relationships. In fact, sometimes those relationships are reversed—adults often learn from kids. The first time Steven (man, age 26) connected to MOOSE Crossing, Pete (boy, age 11) immediately teleported over to say hi and offer help:

```
Pete says, 'hi!'
Steven says, 'hi Pete'
Steven says, 'this is my first time here.'
Pete says, 'okay! want help'
Steven says, 'sure. sound great!'
Pete says, 'do you want to make something?'
Steven says, 'okay'
Pete says, 'wait a minute'
Pete drops MOOSE.
Steven says, 'alright'
 Pete says, 'type 'pet moose''
[Steven types: pet moose]
Steven pets the moose.
MOOSE crosses the crossing forth.
Steven says, 'i like it!'
Pete says, 'try typing 'tel north main street'.'
```

“Tel” is short for “teleport.” They both follow Pete’s suggestion, and rematerialize at North Main Street, where they see this description:

```
You're in a quiet residential neighborhood of Our Town. The road here
ends in a circle surrounded by houses of all different colors and
sizes. To make a home for yourself here type 'build'.
Obvious exits: ..down.........Magic Subway Station
               ..south........Main Street

Steven says, 'that was fun'
Pete says, 'what do you think?'
Steven says, 'how do i build a room?'
Pete says, 'type build'
Steven says, 'okay'
```

Steven types “build” and sees this message:

```
A team of mooses and barn owls walk and fly in, carrying saws,
electric drills, and hammers.  They hammer and saw and bang so fast
you can hardly see them!  In seconds, you have a new house.  They
attach a sign with your name over the door.
Exit from North Main Street (#208) to Steven's Room (#1842) via
   {"Steven"} created with id #1815.
Exit from Steven's Room (#1842) to North Main Street (#208) via
   {"out"} created with id #1843.
Your home has been set to this room.
You or anyone else may enter this room by simply typing your name
   here.  You may leave that room by typing 'out'.
An owl riding on a moose's antelers says 'Welcome to the
   neighborhood!'  The animals pick up their tools and hurry off to
   their next job.
```

Pete sees:

```
   Steven builds a new room here.

   Pete says, 'type tel Steven's room'
```

They both do so. They could actually have gotten there by just typing "Steven," but teleporting works as well. They both see:

```
   Steven's Room
   Welcome to your new home! To decorate it, type 'describe here as "A
   pink house with purple windows a green carpet and silver dots on the
   ceiling."'  Unless of course you don't like pink....
   Obvious exits: ..out..........North Main Street

   Steven says, 'that was easy'
   Pete says, 'so, what do you want to build?'
   Steven says, 'how about a pet?'
   Pete says, 'why not a dog?'
   Steven says, 'a dog it is'
    Pete says, 'alright! first type 'create #5 named' and put a name.'

   Steven types: create #5 named ruff
   He sees: You now have ruff with object number #1087 and parent generic
      thing (#5).

   Steven says, 'okay'
   Pete says, 'press the pencil icon'
   Steven says, 'okay. what button do i press then?'
   Pete says, 'well, in object, type the name, click on script, and type
      pet.'
   Steven says, 'okay'
   Steven says, 'i've now got another window open'
   Pete says, 'now type 'on pet this' in it, and press return.'
   Steven says, 'okay'
   Pete says, 'next, try typing 'emote says "arf" ' and press return'
   Steven says, 'got it'
   Pete says, 'now, type 'end', press save, and close that.'
   From Steven: ruff says arf
```

Austina logs on, pages Steven and Pete hello individually, and then joins them.

```
   Austina teleports in.
   Austina says, 'hi!  Nice to meet you!'
   Pete says, 'hi'
   Pete says, 'try petting ruff'
   Austina smiles
   Austina types: pet ruff
   ruff says arf
   Austina grins.  "Nice dog!"
```

This interaction is entirely typical. Pete helps another adult, Nicole (woman, age 29), later that same day. Similarly, Hermes (boy, age 9) helps Jamal (man, age 18) not long after.

It’s remarkable to watch an eleven-year-old teaching a twenty-six-year-old. When I visited with children participating in an educational MUD called MariMUSE (now called Pueblo) at the Longview School in Phoenix, Arizona in March 1994, I asked a group of children what they liked best about it. One child replied that getting to tell adults how things work was his favorite part. The reversal of normal roles is a powerful experience for many children. Such role reversals are often observed not just in MUDs but in children working with computers in general.

The role reversal is most powerful when the child and adult are in the same room. The kids on MariMUSE were particularly pleased by being able to help their teachers with technical problems. Interacting with others online, the role reversal seems less strange, and in fact may not even be noticed. That is what happened in this particular instance for Pete. He was not even aware that Steven is an adult. When asked about it later via MOOmail (email internal to MOOSE Crossing), he replied:

```
Hi! Anyway, no I did'nt know Steven was an adult. It seemed like
talking to anyone else on moose crossing. well, bye!
```

Pete hadn’t given the matter any thought one way or the other. Steven, on the other hand, thought about it a great deal, but came to no conclusion. It’s easy to find out whether someone on MOOSE Crossing is a kid or an adult, but neither Steven nor Pete was aware of this fact at the time, and neither chose to ask the other. When asked via email how old he thought Pete was, Steven replied:

```
You know, I was curious about that very fact. I really have no idea!
At times I thought he was a grad student at the Media Lab - the way
he instructed me on how to create my room and my pet, the way he
checked my progress. But at other times, I thought he might just be 8
or 10 years old - when I had more specific programming questions, he
didn't seem to understand.

Another strange issue was the duration of time (long pauses) between
Pete’s replies to my questions. If he was a grad student, maybe he
was working on something else at the same time? If he was a kid, was
he working on homework? Having trouble typing? Building objects?
Maybe the server was slow?

Recently I've been trying to learn as much as I can about programming
in MooseCrossings (hanging out in the treehouse) so *I* can actually
help the kids (instead of the other way around).

And one final note, I'm coming to think that the most important
aspects of MooseCrossings are the code of conduct and the
teacher/student relationship that develops between players.
```

A cartoon that appeared in the New Yorker in the early 1990s showed a dog sitting at a keyboard, and bore the caption "on the Internet, no one knows you're a dog." It became an instant classic, and an instant cliché. It's also true that on the Internet, no one may guess that you're eleven. This is particularly beneficial for children whose intellectual level deviates significantly from the level expected for their age. You don't appear slow—people just guess that you're younger. You don't appear frighteningly precocious-—you just seem older, and have the opportunity to socialize with older children on a more equal basis than is possible face to face. This is the case for Hermes (boy, age 9), a boy who is so exceptionally bright that people often react to him oddly. As a fourth grader, he is already taking math classes at a local university. His father Howard (who is also an active MOOSE Crossing member) comments that:

> The nice thing about MC is that it's an outlet for his social interests as well as his writing skills (and his logical/mathematical ability as he gets into programming). Plus it's a perfect father-son activity, especially since he's been much more interested lately in entering into "my" world -- the Net, programming, writing, etc.... Even at a gifted school he has learned to be wary of adults' reactions to him. There's often a strong undercurrent of discomfort/disapproval when he talks or acts above his age-range, and he is old enough to sense it and to start looking for places & activities where age is less of an issue.

On the Net, Hermes can interact with others more free of prejudice than he can in real life. It's worth noting that the invisibility of age online causes problems as well as bringing benefits—on the first MUD Hermes tried before finding MOOSE Crossing, he was almost immediately propositioned. The MOOSE Crossing Code of Conduct strives to remedy that problem. Howard was delighted to find a place for Hermes that is safer.

Returning to Pete's encounter with Steve, two other things are noteworthy. First, the entire interaction—from the time Pete greeted Steven through the time Steven had built a room, made a dog, and programmed his first script—took a total of eighteen minutes. Second, this was only Pete's second day on MOOSE Crossing. He had just learned everything he taught Steven the previous day. Sometimes the best teachers are not experts, but learners only one step ahead of you who are excited about sharing what they themselves just learned.

In addition to learning from people who just learned the same thing recently, members of MOOSE Crossing also seek advice from locally recognized experts. Generally recognized experts include myself, MIT undergraduate Austina Vainius (who has been working on the MOOSE Crossing project through MIT's Undergraduate Research Opportunities Program or "UROP"), and Rachael. It's interesting to note that the ranks of the recognized experts on MOOSE Crossing has come to include one of the older children; it is not limited to adults. Experts are consulted whenever a difficult question arises. Support from experts is complementary to support from peers. Some children are reluctant to ask an expert for help. Once when Rufus (boy, age 12) didn't know the answer to a question Christopher (boy, age 7) asked him, Rufus asked Christopher, "maybe as in I'l ask the next ranger I see for some help or no way?" Unlike Pete who was unaware and uninterested in whether someone is an adult, Rufus clearly notices this fact and treats adults and peers differently. He will rarely ask for help from an adult, and sometimes won't accept it even when it is offered. For these children, the availability of peer support is particularly beneficial. Other children are eager to interact with the experts as much as possible. Still others like Pete are blissfully unaware of who are experts and who are not. A child's attitude towards authority figures is a deeply personal, psychological issue. Having a diversity of sources of help on MOOSE Crossing helps to meet the needs of people with varied personal styles.

Liza (girl, age 10-11) has a more balanced view. Liza is one of the children who comes to the Media Lab each week for the MOOSE Crossing after-school program. I interviewed her on video tape, and asked:

> Amy: How is it different getting help from another kid versus from an adult?
>
> Liza: It depends on who the kid is and who the adult is.... It's not very different.... Some adults just do all the work for you. Some adults let you do the work. Some kids do all the work for you. Some kids just tell you and leave. Some kids stay around and help you more and socialize.
>
> Amy: Which kind do you like best?
>
> Liza: I like the socializing kind better than just telling you what to do. It's much better!

MOOSE Crossing is particularly conducive to the "staying around and socializing" sort of help that Liza prefers.

Since experts are authority figures, people are often eager to show them completed projects. The design of MOOSE Crossing deliberately attempts to reduce the dominance of adult authority—it's not a place where you need to impress the teacher. However, many kids still take pleasure in asking for approval from authority figures.

Barbara Rogoff contrasts three models of learning: adult-run (where adults transfer information to children), child-run (where children discover things on their own, with adult support only when requested), and a community of learners (where everyone is participating in shared activities, and learning is a process of transformation of participation). Newcomers to a community-of-learners approach may not immediately understand the nature of the learning taking place there. In a school using this model at which Rogoff was a participant-observer, parents (called "co-opers") volunteer three hours per week at the school. New co-opers often don't initially understand the activity going on, because they're trying to interpret the activity going on there in terms of either child-run or adult-run models of instruction. Rogoff writes that:

> In fact, one indicator of alignment with the philosophy of a community of learners in a school seems to be regarding oneself as a learner, continually. Adults who have a learning attitude find, as one co-oper reported, "one of the things that's been nice for me has been having kids, first graders, come time and say 'don't worry about it, I'm gonna show you how to do it!'" (Rogoff 1994)

The reversal of typical roles can be energizing for adults as well as kids. For Rogoff, it's an important feature of such environments that both adults and children see themselves as learners. On MOOSE Crossing, I originally anticipated that adults would volunteer time to help the children. In fact, the children more often help the adults. The interaction between Pete and Steven is typical. Most kids have more time to devote to MOOSE Crossing than adults and more genuine interest, and therefore have greater expertise. Almost all members of MOOSE Crossing of all ages have at some point taken on a teaching role, and that is an important part of the learning experience.

A wide variety of factors have come into play in this discussion of technical and emotional support, including age, attitude towards authority, and the nature of the personal relationship between the helper and the helpee. Asking for, receiving, offering, and providing help are not simply exchanges of information. They are social acts that take place in the context of networks of relationships. Community structures that reorient the typical patterns of those relationships can change people's feelings about giving and receiving help. When I asked people on MOOSE Crossing's "social-issues" mailing list how getting help is different on MOOSE Crossing versus in school, Rachael replied:

```
From:     Rachael
To:       *Social Issues
One thing I'd like to mention about helping other people here, is
that just about everyone I've met at least seems like they _want_ to
help you, and aren't just obligated. They don't seem upset about
helping you out. That's different from school, when a lot of people
feel kinda upset when they help you.
```

```
Also, it is a lot easier here. Most people (including myself) will
drop all but the most important projects or at least set up a date
when someone needs help.

Rachael
```

A community of students in a traditional classroom have a different set of social relationships than the community of MOOSE Crossing. In some cases, students may be reluctant to ask for help because they know the teacher is grading their performance. In other cases, a teacher may be burdened with having too many students and not have enough time to help each student individually. A particularly harried teacher might slip into a brusque tone of voice when demands are made on him or her. These are just stereotypes—every classroom is a unique community with its own patterns of interaction. The broader point is that typical patterns of interaction among community members affect how comfortable each participant is in asking for help or offering it. In turn, the patterns of asking for help affect other patterns of interaction of the community. On MOOSE Crossing, these have become mutually positively reinforcing. People generally receive supportive, friendly help, and then are eager to offer it to others. A friendly tone to the place makes people comfortable asking for help, and the ready availability of cheerful help makes the place seem more friendly.

## 5.6 An Appreciative Audience

Kids are always showing off their projects to others on MOOSE Crossing—often to everyone they meet. For example, one Saturday in October 1996, Hermes was logged on for much of the day. He worked all day on improving his "generic magician," a character class that lets you cast spells. At this time, he was using one of his alternate personas, Nick. On meeting a new member, Jamal (man, age 18), Nick/Hermes immediately showed off what his new scripts can do:

```
Jamal says, 'hi'
Nick types: cc mr
Nick casts a mirror spell!
Everything that touches Nick bounces back as if something was
   shooting at it!
You try to touch him but you feel a LARGE shock and your hand
   fliesbackward as quickly as your arm allows!
Nick says, 'like it?'
Nick grins
Jamal says, 'pretty cool.'
Nick types: cc fb
Nick casts a fireball spell!
The fireball starts a fire that lights up the room!

Jamal types: look Nick
He sees:
```

```
is alot like hermes exept he's a little bit taller and a little bit
stronger.He has a peice of rusty metal in his hand. It has a strong
burning aroma and every once - in - a - while it zaps out red beams
of light that singe everything they touch.
He is awake and looks alert.
Carrying:
generic magician

Nick says, 'i've got 4 more'
Nick casts a fgjhgfhjg spell!
Jamal says, 'a what spell?'
Nick says, 'mad typing'
```

Nick goes west, and Jamal sees him leave, followed by his magic harp and pet human Google. Nick pages him “come on, it's west,” and Jamal follows. Having shown Jamal a few of his spells, he now wants to show Jamal another one of his projects, his magic subway system. Jamal sees this room description:

```
The Station Master's Office
The office of Hermes, the Magic Subway's Station Master.   It looks
like a bubble surrounded by coral that is floating through time and
space itself.  There is very little furniture...a bed, something that
looks like a huge blob of jello that might be a couch, and a
chandelier that has small glowing fish hanging from it.  They look
like they don't give a darn that they're hanging upsidedown...in fact
they look rather happy up there.
   Obvious exits: ..east.........Closet
                  ..down.........Magic Subway Station
You see Magic School Bus, Magic Harp, and Google here.

Jamal types: look Google
He sees:
A short, brown, shaggy haired humanoid.  He is very tough and strong
for his size.  He has a short-sword at his belt.  He will protect
himself or his owner with it.

Nick says, 'like my room?'
Jamal says, 'The Closet?'

Jamal types: look Magic Harp
He sees:
You see a brand new following object that needs a description!  Type
'help #342' for more information

Nick says, 'what about it?'
Nick says, 'no. my room is here'
Jamal says, 'The Closet's your room?'
Nick says, 'no.here is.your in it'
Jamal says, 'You're the Station Master?'
Nick says, 'yep'
Nick types: become Hermes
Nick freezes momentarily.  You feel a psychic wrench as if a great
   spell is being broken.  The figure before you dissolves into a
   normal human boy.  Hermes is back!
Jamal says, 'Cool. Looks like I'm meeting an important person.'
Hermes says, 'that answer your question'
```

```
Jamal says, 'yeah.'
Hermes says, 'yeah. i made this hole thing...with my dads help'
Jamal says, 'It's pretty impressive.'
Hermes says, 'wanna be a mgician?'
Hermes types: oo magician
One of Hermes's harp strings breaks with a loud twang!
He must have made a mistake.
Hermes hits himself on the forehead and says " D'oh!"
Hermes says, 'I meant 'magician'!'
Jamal says, 'Don't know if I'm qualified. I'm new to all of this.'
Hermes says, 'you don't have to be qualified'
Jamal says, 'okay then.'
```

The interaction began with Nick/Hermes showing off his creations to a newcomer, Jamal. Jamal's appreciation gave Hermes positive reinforcement. The encounter has now made a smooth, natural transition to a tutoring session where Hermes is helping Jamal.

```
Hermes says, 'type parents Hermes'

Jamal types: parents Hermes
He sees:
Hermes(#1354)   generic magician(#1448)   Generic Multiple-
Personality Character(#456)   MOOSE player class(#108)   generic
programmer(#59)  generic builder(#4)   Frand's modified player
class(#141)   generic player(#6)   Root Class(#1)
Hermes also looks at his own parents, and sees the same thing.

Jamal says, 'okay. I see a list of objects.'

Hermes types: parents Jamal
He sees:
Jamal(#1599)   MOOSE player class(#108)   generic programmer(#59)
generic builder(#4)   Frand's modified player class(#141)   generic
player(#6)   Root Class(#1)

Hermes says, 'now type chparent me to gm'

Jamal types: chparent me to gm
He sees:
You must give the name of some object.

Jamal says, 'It says I must give the name of some object.'
Jamal looks at Hermes' parents again.
Hermes says, 'oooook.type chparent me to generic magician'
Jamal tries this, and gets the same response.
Hermes says, 'or #1448'

Jamal types: chparent Jamal to #1448
He sees:
Rebuilding Jamal (#1599)
Done.  1 objects rebuilt.
Parent changed.
```

You can’t refer to an object by its name unless it is in the same room as you or you are holding it. Jamal needed to refer to generic magician by its number. Hermes has learned something from teaching Jamal. Later in the day, he will similarly help Pete to become a magician. When he helps Pete, he will get the command right the first time.

```
Jamal says, 'done.'
Jamal checks his own parents and Hermes’ parents, and sees that they’re the same.
Hermes says, 'ok. type cc fb'
Jamal types: cc fb
Jamal casts a fireball spell!
The fireball starts a fire that lights up the room!
Jamal swrls his cloak and smothers the fire!
Hermes says, 'see!?'
Jamal says, 'cool!'
Jamal says, 'thanks.'
Hermes types: cc mr
Hermes casts a mirror spell!
Everything that touches Hermes bounces back as if something was
   shooting at it!
You try to touch him but you feel a LARGE shock and your hand
   fliesbackward as quickly as your arm allows!
Hermes types: cc fb at jamal
Hermes casts a fireball spell at Jamal!
The fireball covers Jamal in red and orange flames!
Hermes swirlshis cloak and smothers the fire!
Hermes types: cc bq at jamal
Hermes covers Jamal with a bright light and sends him to Mars and
   yells after him 'BE QUIET!'
he thinks hard as soon as he hears a tiny 'All right' and brings
   Jamal back to earth.
Hermes types: cc nv
Hermes casts a light spell!
A small glowing sphere appears in his hand.
he see's what he is looking for and turns the light out.
Hermes types: cc armor
Hermes casts a armor spell!
Jamal says, 'You've got a lot of spells'
Hermes says, '6 infact'
Hermes types: cc mr
Hermes casts a mirror spell!
Everything that touches Hermes bounces back as if something was
   shooting at it!
You try to touch him but you feel a LARGE shock and your hand
   fliesbackward as quickly as your arm allows!
Jamal says, 'So what does the Station Master do?'
Jamal types: cc
Jamal casts a  spell!
Hermes says, 'oh...fix the subway when it needs it'
Hermes types:  cc
Hermes casts a  spell!
Hermes says, 'neat!'
Hermes says, 'never thought of that!'
Jamal says, 'was an accident actually.'
Hermes says, 'oh.still ....it suprises people'
```

Jamal accidentally cast a spell with no name. Hermes decides this is a clever trick, and compliments him on it.

```
Hermes says, 'let's go to the subway'
Jamal says, 'okay'
```

They both go down, continuing their tour of Hermes' subway system. They see:

```
Magic Subway Station
Welcome to The Magic Subway Station!  You are deep underground, in a
great hall carved out of solid rock.  You see a regular stream of
travellers emerging from the shimmering magical entrances that line
the west side of the hall.  Most hurry across the room and vanish
into the magical exits on the opposite side, but a few stay to admire
the magnificent interior.  (Typw 'view hints' for suggestions.)

When you are ready to leave, please select your destination and step
   through the magic portal!
  Obvious exits: ..CR...........Crossroads
                 ..EA...........Emerald Apartments
                 ..HC...........Home in the Clouds
                 ..IC...........Mouse's favorite Ice Cream
                 ..NM...........North Main Street
                 ..PI...........Paradise Island
                 ..RF...........Redwood Forest
                 ..ST...........SparkyTown
                 ..TA...........Travel Agency
                 ..XX...........The Station Master's Office

Hermes says, 'type view mural'
```

Jamal types: `view mural`
He sees:

```
At the far north end of the hall, you see  a gigantic three-part
mural, showing the construction of the Subway Station.  The three
parts are called "PLANNING," "CREATION," and "COMPLETION."

Hermes says, 'or view hints'
```

Jamal types: `view hints`
Hermes types the same thing.
They see:

```
A luminous scroll dances through the air towards you and unrolls
before your eyes!  It says:  "You can view north, south, east, west,
up, down, and other things mentioned in those views.  In addition,
you particularly notice a statue in the center of the hall."  The
scroll rolls itself up with a saucy snap and dances away, humming a
spritely tune!
```

Jamal types: `view mural planning`
He sees:

```
There is nothing to view by that name!
```

Jamal types: `view "PLANNING"`
He sees:

```
In the left-hand portion of the mural you see a slim boy with tousled
brown hair.  He is standing in a rough natural cavern, surveying its
walls and ceiling by magical light.  He holds a slender rowan wand in
one hand, and a small singer's harp rests at his feet.  The wand and
the harp shimmer with a powerful magical aura.  There are two more
sections of the mural in the center and on the right.
```

Hermes types “v n” and sees the same introductory description Jamal saw when he viewed the mural.

Jamal types: `view center`
He sees:

```
In the foreground of the center part of the mural you see a slim hand
holding a wand.  Powerful lightning streams from the wand in a great
arc.  Ahead of the lightning are the rough rocky walls of a natural
cavern, but where the lightning has already passed you see the walls
of the present Subway Station taking shape.  There are two more
sections of the mural to the left and right.

Hermes says, 'type v planning'
```

Jamal types: `view right`

```
He sees:
In the center of the rightmost part of the mural a boy is using his
wand to create great magical gates and link them to faraway
locations.  In the background is a shadowy older figure, using his
staff to carve columns and create chandeliers and other decorations.
```

Hermes types “v planning” and sees the description Jamal saw earlier.

```
Jamal says, 'cool. it's you in the mural right?'
Hermes says, 'yeah.how'd you guess?'
Jamal says, 'the wand and the harp.'
```

Hermes types “v creation” and sees the same description Jamal saw on the center wall. Then he types “v completion”, and sees the same description Jamal saw on the right wall.

Jamal types: `look CR`
He sees:

```
A shimmering gateway of pure energy, set into the east wall of the
Subway Station.  It leads to the Crossroads.
```

Jamal types: `look HC`
He sees:

```
A shimmering gateway of pure energy, set into the east wall of the
Subway Station.  It leads to the Home in the Clouds.

Hermes says, 'type v completion'
```

Jamal types “v completion” and sees the same description he saw on the right wall.

```
Hermes says, 'the shadowy guy is my dad'
Jamal says, 'I guessed he was.'
```

Hermes types: `v statue`
He sees:

```
In the exact center of the Subway Station, you see a heroic statue of
a large moose, standing on his hind legs.  He's wearing glasses and
agoofy grin, and his antlers seem a bit crooked.  He appears to be
```

```
staring off into the distance.  In one hand, he is holding a model
sailboat that reflects deep red sparkles.  There's a flying squirrel
sitting on the moose's shoulder, wearing goggles and a leather flying
helmet.

Hermes says, 'type v statue.wonder who they are?'
Jamal looks at the statue, and sees its description.
Jamal says, 'Is it Rocky And Bullwinkle?'
Hermes says, 'yeah!'
Hermes goes ding!ding!ding!
Jamal says, 'Why are you dinging?'
Hermes types: kick google
Hermes kicks Google hard in the stomach!  Google lands on his butt,
   but bounces right back up!
Google says, Whadya do dat for, ya big louse?!!
Google then pulls out his sword and threatens to hack Hermes into
   bits.
Hermes says, 'game shows do that when you get an answer rite'
Jamal says, 'I see. Thanks... Go easy on Google. Seems kind of
   feisty.'
Hermes says, 'yeah... i will'
Hermes says, 'hey! want me to put a create candy spell?'
Jamal says, 'Sure.'
Hermes says, 'on us that is'
Jamal says, 'As long as it doesn't turn us into candy.'
Hermes says, 'ok. just a sec.click on the pencil, type me in the
   browser and duble click on cast'
```

Having an audience has inspired Hermes to continue to improve his set of spells. He adds a new spell to his magician. Jamal waits a few minutes, checking help messages.

```
Hermes says, 'now type cc cc'
Jamal types: cc cc
Jamal snaps his fingers and your favorite kind of candy appears in
   him hand!
Your mouth waters at the sight of that candy!Yum!Yum!
Jamal says, 'That's pretty cool.'
```

Hermes is unhappy with the incorrect pronouns in the first line of the script's output. He modifies the script and tries it again. The script prints out:

```
Hermes + my pp + snaps your favorite kind of candy appears in + my pp
   + hand!"
Your mouth waters at the sight of that candy!Yum!Yum!
```

There's still an error. Now a mistake in the placement of quotation marks is causing a line of the script not to be correctly evaluated. Hermes modifies the script and tries it again. This time it works:

```
Hermes snaps his fingers and your favorite kind of candy appears in
   his hand!
Your mouth waters at the sight of that candy!Yum!Yum!
```

Here’s the final version of the candy script:

```
on cast "cc"
  if context isn't me
    return
  endif
  announce_all my name + " snaps " + my pp + " fingers and your
   favorite kind of candy appears in " + my pp + " hand!"
  announce_all "Your mouth waters at the sight of that candy!  You
   take the candy and gobble it up!  Yum! Yum!"
end
```

Hermes is enjoying being the teacher:

```
Hermes says, 'yeah.hope you can follow my lead'
Hermes says, 'so you can make your own spells'
Hermes types: cc bq
Hermes casts a bq spell!
Hermes says, 'nuts!'
Hermes says, 'it doesn't work'
Jamal says, 'I think I can follow your lead. I have to go soon.   But
   I'll make a spell for next time I'm around.'
Jamal says, 'Did you create the generic magician?'
Hermes says, 'yeah. if you have any ideas for spells tell me.by mail
   or talk'
Jamal says, 'Okay.'
Jamal says, 'Thanks a lot for your help.'
Hermes says, 'don't try cc bq!'
Hermes says, 'it won't work easily'
Hermes says, 'you'ljust get a turkey leg of embarresment!'
Jamal tries to smile, but can’t figure out the command.
Jamal says, 'Thanks for the tip.'
Hermes says, 'welcome'
Jamal says, 'I'll talk to you later. I have to go.'
Hermes says, 'by'
Hermes says, 'wait'
Jamal says, 'Yeah?'
Hermes says, 'i'll tell you when it's okay to do cc bq ok?'
Jamal says, 'Okay.'
Hermes says, 'by'
```

Jamal, an MIT undergraduate studying computer science (or “Course 6” in MIT lingo), was impressed with his encounter with Hermes. Asked via email about his impressions of the encounter, he replied:

> I remember meeting Hermes. The deepest impressions left on me from our conversation were his knowledge of programming in Moose Crossing and the ease with which he was able to teach me about what he'd done and what he knew. I even talked to one of my friends later that day about him. It seemed then (and still does) to me that if he continues along his present course he'll be well-prepared for Course 6 by the time he's old enough.

> From what I can tell your MOOSE Crossing MUD is doing exactly what you said it would; allowing kids (and others) to learn by doing and teach others. I think it's great.

During the course of the day, Hermes showed his spells off to three different people: Jamal, Pete, and Miranda. He showed Jamal and Pete how to change their parent object to his character class so they can cast spells too, using his spell scripts. Miranda already had her own character class, which has an answering machine which records things that you hear when you're not connected. Hermes changes his character class to inherit from hers, so people can use both at once.

Whenever anyone finishes a project on MOOSE Crossing, they almost always rush to show it off to their online friends. Positive reinforcement is always available. In fact, while I have been writing this paragraph, Mouse asked me to come see the pool she is building as part of her hotel, and Miranda showed me the program she wrote to be able to change her facial expression/mood separate from changing her description. (I often have a MOOSE Crossing session open in another window while I write.) The availability of a potentially large audience helps motivate people to create things.

Another way kids show off their creations to one another is by placing advertisements in our online newspaper, the MOOSE Herald-Examiner. Here is the advertisements section on the day of this writing:

```
*** ADVERTISEMENTS ***

JACKET -- 10/14/96
~~~!~~~!~~!A COZY JACKET TO SLEEP IN!~~~!~~~!~~~
MOUSE HAS MADE A COZY JACKET! SOUNDS WEIRD BUT IS A GREAT PLACE TO
BUILD A HOME OR A STUDY OR MABEY JUST A ROOM! JUST TEL TO THE JACKET
AND SEE WHAT YOU CAN DO ON A COZY JACKET! IF YOU HAVE ANY QUESTIONS
OR COMENTS PLEASE MAIL MOUSE OR TALK TO HER IN PERSON. THIS IS A
GREAT PLACE TO BUILD ATTICS OR OTHER PLACES WITH LITTLE NOCKS.
REMEMBER: AFTER A MOUTH AND MOUSE DOES NOT DO WELL WITH THE JACKET,
THE WHOLE THING WILL BE RECYCLED! -- Mouse

POTATO -- 10/9/96
I just made a cool potato! You can plant new potatos with it. You can
also mash it, french fry it, make potato chips, and more! When you
use those scripts, it changes the potato's appearance! You can also
fix the potato after you've cooked it. Come see it! -- Miranda

MOUTHPIECE -- 10/09/96
Now, if you want to talk to a bunch of people at once who are in
different rooms, then join mouthpiece (#1684) by typing @addfeature
#1684, and then turn it on by typing "on mp" . Then, you can talk by
typing 'shout <whatever> into mp'. Mouthpiece is public, so anyone
can join. If you want to start your own, then make one...mouthpiece
is also a generic! -- Rachael
```

```
HAVE YOU EVER BEEN ON MOOSE CROSSING WITH NOBODY TO TALK TO? --
9/30/96
Now, whenever you feel like it, you can talk to your
generic_conversation object! With this object, you can ask it yes or
no questions, and whenever you tell it something, it will answer you!
Starting with asking you how you're feeling, this objct will talk to
you for hours on end, and never have to disconnect!  -- Miranda

MOOSE MAGIC -- 9/30/96
~~~~~~~~~~~~NEW MOOSE MAGIC~~~~~~~~~~~~
Ever wish you could du extraordinary stuff, like make people float,
or shoot out fake fireballs? Impress your friends by getting a free
spellbook at the Owl's Nest. Good for magic fights at the SparkArena.
To even further these powers, get an owl, all at the Owl's Nest. You
must find the Owl's Nest at your own risk. -- Rufus

UTOPIA -- 9/23/96
Do you need a break ? well if you do come to UTOPIA the greatest
resort in the land.it has swimming,shop, and a realy cool arcad
come now to UTOPIA . this lovely resort is in red wood forest off
liza s room.(for any more info talk to liza) -- liza

ANSWERING MACHINE -- 9/23/96
I just finished a character class which is an Answering Machine! It
records what people tell you when you're asleep! For more details
mail Miranda.
P.S. Please mail me if you would like to add a room to the travel
agency! -- Miranda
```

The power of having an audience can be seen clearly in the explosion of interest in the World Wide Web in 1995-1996. People's home pages are a form of multimedia self portrait. The tools to make such self portraits (for example, paint programs) have been commonly available (to middle-class Westerners) since the mid to late 1980s. However, it was only when tools to *share* such self portraits with a large audience became commonly available that interest in making them took off (Bruckman 1995).

MOOSE objects, like home pages, help to establish an individual's identity within the community. The first interaction between two members often focuses on admiring one another's creations. With most children being followed by a small entourage of pets, it's easy to see how strongly one's creations affect one's public image. Notice that without being prompted, Jamal examined Google and the magic harp following Hermes. This is typical.

More subtly, people’s creations also affect one's self image—deciding what sort of a thing to make is partly a reflection on the question "what sort of a person am I?" Goffman notes the role that clothes play in defining one's sense of self and role within the community (Goffman 1959). Appadurai studied the same function of objects collected in people's homes (Appadurai 1986). Sherry Turkle takes this analysis a step further, noting how children use computational media to work through deeply personal issues—things you make are even more expressive of who you are than things you choose to wear or buy. Computational media present a particularly rich opportunity for expression, because they can be used to make such a diversity of meanings. The computer is a kind of Rorschach Test (Turkle 1984). Children not only express who they are but also help to shape who they are through their construction activities.

These presentations of self are motivated and shaped by the existence of an audience. Making an object on MOOSE Crossing is a fundamentally social act which exists in relation to an audience. Rachael summed it up in the quote that opens this chapter: "While programing is a lot of fun, I don't think I'd do it, if there wasn't anyone who would apprecitate it."

## 5.7 Local Community and Online Community

So far in this chapter I have been focusing primarily on the impact of the *online* community on the participants in MOOSE Crossing. The children's experiences are of course also affected by the local, face to face community they are a part of while they participate. The local setting varies. Most children participate from home; however, that may mean they are primarily alone, or are interacting with parents or siblings. Some children participate in classes at school. Some participate in after-school programs. The group I have studied most closely are the children who came to the after-school program I ran at the Media Lab.

On average six children (of the roughly 160 children registered as of March 1997) came to the Media Lab once per week for more than a year to participate in a MOOSE Crossing after-school program. Each session lasted roughly two hours. The exact composition of the group changed slightly over time—older children started bringing younger siblings, and one pair of siblings dropped out.[3] (The reasons for their loss of interest are not clear. MOOSE Crossing,

[3]First, just Miranda (girl, age 10-12) came. She was the first child to work with the MOOSE software. The software was modified and bugs fixed based on her comments and comments of other children to join later. Then her friend Byron (boy, age 10-11) joined. Zoro (boy, age 11-12) joined the group next. The entire time Miranda was participating, her younger sister Mouse (girl, age 8-9) had been watching quietly over her shoulder. Miranda's mother hadn't had a place to leave Mouse while Miranda was at the Media Lab. After several weeks of observing, Mouse decided she was ready to try it too. Mouse's best friend at school is Byron's younger sister Patricia (girl, age 8-9). (In fact, the older children met through the friendship of their younger siblings.) Mouse enthusiastically invited Patricia to join. Patricia never really shared Mouse's enthusiasm, and only came for a few weeks. Approximately a month after she stopped coming, her older brother Byron stopped coming as well. At around that time, Zoro's family decided to send their second child Liza (girl, age 10-11). Next, Miranda invited her best friend from school, Squirrel (girl, age 11) to join. Next, Zoro's family started sending their third child, SpaceRabbit (boy, age 9). Their youngest child Goofy (girl, age 7) joined the group towards the end of the year.

like any activity, is not for everyone.) The size of the group was limited by the number of Macintosh computers we had available for them to use.

All of the children in the after-school program had access only from the Media Lab for the first half of the year. Mid-way through, Miranda and Mouse got access from home as well. Towards the end of the year, I asked both girls to compare using MOOSE Crossing at home versus at The Media Lab. They both said that it's easier to get their questions answered when they are using MOOSE Crossing at the Media Lab than at home. It's easier to ask a question out loud than type it. Additionally, they mentioned that Austina and I always know the correct answers; other children online sometimes do not. However, Mouse mentioned that getting help from Rachael is just as good as getting help from adults—she really knows what she's doing.

One notable difference between home and classroom or after-school program use is the level of activity in the room. The weekly sessions at the Media Lab often get boisterous. One teacher using MOOSE Crossing in her classroom in California called the experience "magical chaos." While it may be easier to get help in an after-school program, Miranda commented that it's quieter when she's working at home, and she can focus more. In contrast, Mouse complained that at home "It's hard to type because Miranda is always there saying 'I want to get on! I want to get on!'" The girls share one computer at home, and the older sibling seems to win control of it more often. At the Media Lab, each girl gets her own machine. The details of the local setting can be substantially different even for two kids using the same computer in the same family setting. That local setting strongly effects a child's experiences.

My impression based on observing Miranda and Mouse over the past year and a half is that having access at home significantly enhanced their enjoyment and enhanced the benefits they got from the project. Children who can participate only once a week may lose interest in a project between weeks. When Zoro (boy, age 11-12) arrives at the Media Lab, he often asks to be reminded of what he was working on the previous week. In contrast, Miranda and Mouse often log on again immediately on arriving home and continue working on their project. When they are excited about a project, they may work on it every day.

The increased rate of progress I observed in Miranda and Mouse after they got access from home can obviously be substantially attributed to their increased time on task. However, there are also other more subtle factors:

- **Engaging with others online**

  When they are at the Media Lab, Mouse and Miranda (and the other children) talk primarily to people in the room, and have fewer conversations online. It's only when they go home that they really start to engage with other children online. Talking with other children from far-off places is good practice for written expression, may open new cultural perspectives, and helps to form a connection to the broader community of members rather than just the local community of their after-school program.

- **Teaching**

  When Miranda and Mouse are at the Media Lab, they primarily work on their own projects and leave it to the adults to help the other children. When they are at home, they often take the time to teach others (as we shall see Miranda do in the next section). Teaching is often as much an educational experience for the teacher as the learner.

- **Invisibility of some social factors**

  Another issue concerns the invisibility of a variety of social factors online. The invisibility of age was discussed in Section 5.5. When Mouse is at the Media Lab, she is usually the youngest child present, and gets treated as such by her peers. Online, other children often guess that she is much older, because of her greater experience with the system. (She told me with pride that Pete (boy, age 11) thought that Mouse was 12). Other factors in addition to age that can sometimes be beneficially invisible online include gender, physical attractiveness, popularity, social class, and having many kinds of handicaps.

- **Spontaneous versus scheduled**

  Finally, the home and after-school program environments differ in how they are scheduled. At home the girls connect only when they are in the mood to do so, and therefore well disposed to benefit from it. The group at the Media Lab meets at a regularly scheduled time. Sessions can be less productive if the child happens at that time to be tired, hungry, sick, or just not in the mood.

Benefits of having access in a more formal group setting (i.e. classroom or after-school program) include:

- **Ease of communication out loud versus in text**

  For most people, speaking is faster and requires less effort than typing. It’s easier to ask a question out loud. Putting a question into written words usually requires more reflection. This reflection in some cases may help the questioner to understand the question better; however, the effort required can often be prohibitive, particularly for younger children. Other channels of communication such as body language and intonation can also be useful in getting across a meaning.

- **Someone helping you can see your screen**

  It's much easier to help someone if you can see exactly what they're seeing. It's possible in theory to solve this problem in software by letting one person have a view of exactly what another person's screen looks like. There was not enough time to implement this feature for the MOOSE Crossing project. Consequently, it's easier to help someone who is in the same room.

- **Presence of experts**

  If the community is large enough, there will always be "expert" help available online for difficult questions. MOOSE Crossing is still small enough that this is not always the case. In many group programs organized to date, the adult leaders of those programs have not had the time to develop even a basic understanding of the software. This is understandable—most teachers are overworked. Children in such programs have much more difficulty compared to those in programs where experts are present. Over time, some of the children may develop significant expertise, and be able to step into the expert role. One teacher writes that "We are learning together and I am learning from them as much or more than they are than they are learning from me. The bolder children have a sixth sense about stuff like this and forge ahead. They are becoming all of our tutors." They started out with little local expertise, but developed more over time. Having experts present is a significant advantage.

- **Scheduled vs. spontaneous**

  The advantages of spontaneous time dedicated to an activity were discussed above. Scheduled time also has significant advantages. Time in a formal program has been set aside for that purpose, and is usually not interrupted. Other activities are not competing for the child's attention during that time. Setting aside a regular time for an activity also means that a child is more likely to finish the difficult or boring part of a project. It's natural to set aside a project for a while when you get tired of it, or get mired in a difficult or boring part of it. While a child may never return to a project stuck in such a state if he or she is working on it spontaneously from home, he or she is likely to return to it during the next of a set of regularly scheduled meetings.

A combination of spontaneous and scheduled time has significant advantages. In most of the classrooms in which MOOSE Crossing is currently being used as an organized activity, the children have access once or twice per week during a regularly scheduled time. However, in one notable school, the computers are located in the classroom, and children may use them during breaks in the school day. The classroom has one computer for every two to three children. To save space, the computers are recessed into desks—the surface of the desk is Plexiglas, and the monitor is underneath, at an angle. Many of the children chose to use MOOSE before school starts, during recess, and at other free periods. While not all children take advantage of this opportunity, those that do have become local experts. During the class' regularly scheduled sessions, they often help the other children. The class has been able to accomplish much more than other classes. It's not just a matter of increased time on task, but also of giving students the opportunity to chose to become more involved, and cultivate a genuine, self-motivated interest.

The students in this class see the computer as a tool, and use it fluently and naturally for many tasks. To draw a common analogy, pencils wouldn't have had much of an impact on education if students were scheduled to go to the pencil room for an hour once per week. To be effective, tools need to be plentiful and always available—available to use in organized activities, and during free time.

Interaction in an organized program and interaction during free time are complementary. They each have different benefits, and work particularly well together. Similarly, interaction face to face and interaction over the net are also complementary. This is true in a broader sense of online communities in general. My best relationships facilitated by MediaMOO are with people who I see face to face once or twice a year at conferences (scheduled, face to face), and keep in regular contact with on MediaMOO (unscheduled, over the net). Face to face contact is richer, but online contact is easier to maintain on a regular basis.

A related issue is the relationship between geographically local and remote community. Alan Shaw's MUSIC project uses a computer network to help improve communication in an inner city neighborhood (Shaw 1994). An online community can help to cement relationships within a geographically local community. Children in MOOSE Crossing programs benefit both from increased interaction and collaboration with their local classmates, and from interactions with children in other cities and countries who they would not otherwise have met. Local community and virtual community are not in opposition, but rather for most applications work best together.

## 5.8 An Extended Example: Lady's First Script

It's worth tracing issues of community support through another extended example: Lady's first script. Lady (girl, age 11) first connected to MOOSE Crossing on a Friday afternoon. Austina helped her to make a dog named Cutey, by making an object which inherits from generic smart dog. Lady returned the next Monday and explored a bit, trying to strike up a conversation with various people. Finally, she joins Miranda, who is in a room called "lab" owned by Jack (boy, age 13). Their conversation is presented here unedited. It took place over a period of 48 minutes.

Jack's room looks like this:

```
lab
You are in a hidden lab.  You don't see any exits, but you doo see a
beam in the back of the room.  Strange robots and potions are on
desks, and non compiled scripts are floating all over!
You see fluffey here.
```

Here's how the conversation begins:

```
Lady says, 'Hello'
Jack says, 'i made a secret passage :)'
Lady says, 'You did, Wow!'
Miranda says, 'Cool!'
Miranda says, 'Where'
Lady says, 'Yeah, where?'
Miranda smiles
Jack says, 'if you are in my room, just type beam to get here if you
   are here, type beam to get to Jacks room.'
Jack says, 'it is a script.'
Lady says, 'Cool, how did you do that?'
Miranda says, 'How old are you Lady?  I don't think I've met you.
   Sorry I just left after you joined me.'
Lady says, 'I'm new, I'm 11'
Jack says, 'this is the script: on beam'
```

[They all type 'beam' to get to Jack's secret room, and 'beam' again to get back.]

```
Miranda says, 'Hello again'
:)
```

[The smiley face was announced by Miranda. Everyone in the room sees it without being told who generated it. Technically, this is against the code of conduct all members agree to on joining, but most people don't mind if it's done in moderation in good taste.]

```
Jack says, 'hi :) sorry about that'
```

[Jack left to try out the passage. He's apologizing for disappearing suddenly. The others followed him and understood what was going on, so no apology was really necessary.]

```
Lady says, 'hello again to you too'
my personal favorite is :b
```
[announced by Miranda]

```
cute huh?
```
[announced by Miranda]

```
Lady says, 'what is :b'
or :d
```
[Also announced by Miranda. No one ever answers Lady's question. Those are emoticons of someone sticking their tongue out.]

```
Jack says, 'you can see the script by clicking on the pencal and
   typing HERE on the box you get when you click on the pencil. Then
   you click on script, and type beam and hit OK.'
Miranda says, 'ok'
Lady says, 'OK'
Jack says, 'yes :b is cool'
Lady says, 'Wow, but how do you make a script?'
```

Lady and Miranda were just wandering around saying hello to other kids. Jack uses their arrival as an opportunity to show off his new creation, his beam script. He receives immediate positive feedback from his peers. Lady is impressed. Jack's achievement inspires her to want to learn how to make her own script.

```
Miranda says, 'Cool script! It makes a lot of sense.  Did you make it
   yourself? By the way, I'm the one who announced :b'
Miranda says, 'Well do you have any creatures yet?'
Miranda says, 'Lady?'
Lady says, 'yes'
Jack says, 'yes i made it myself :) lets teach Lady some scripting!'
Miranda says, 'If you want to make a script, you go to the pencal,
   and type your objects name'
Lady says, 'what's the pencal?'
Miranda says, 'But first you should know how to make scripts'
Miranda says, 'So don't do what I just said yet.'
Lady says, 'ok'
Miranda says, 'Why don't you look at the help on scipting?'
Lady says, 'how?'
Jack says, 'i have an idea'
Lady says, 'what is it?'
Miranda says, 'You go to MOOSE, on the top of your screen, (next to
   size and windows) and select 'help''
Jack says, 'how about i walk you through making a magic gem!'
Miranda says, 'I'll help!'
Lady says, 'ok'
```

Jack and Miranda approach the task of teaching Lady how to program enthusiastically. They're teaching technique is not, however, ideal. Instead of asking Lady what she wants to make, Jack suggests a project for her: a magic gem. Throughout the conversation, Jack will tell Lady exactly what to type rather than giving her examples and general principles. Miranda, who has tutored others before, is better at explaining the underlying ideas. Still, a trained teacher could do much better. Peer tutoring is not a panacea. Despite the imperfect nature of the tutoring she's receiving, Lady learns how to script well enough to create a similar script on her own the next day. The children are also developing a friendship as they are learning together—and it's not just Lady who is learning. Jack and Miranda are developing a better grasp of the material they are teaching, and are also learning something about teaching itself.

```
Jack says, 'OK. first, create $thing called gem.'
[Lady types "create gem" which doesn't work.]
Miranda whispers to Jack, 'hi'
Jack whispers to Miranda, 'hi'
```
[Whispering lets you communicate with a specific person without everyone else in the room hearing.]
```
Miranda whispers to Jack, 'What's the gem going to do?'
Lady says, 'how'
Jack says, 'type this: CREATE $thing called gem'
```
[Miranda types what Jack was proposing to check it, and then immediately recycles the gem she has made.]
```
Miranda opens a hidden trash chute.
Jack whispers to Miranda, 'I don't know yet!'
Lady says, 'Oh, cool'
Miranda whispers to Jack, 'how about it sparkles when someone touches
   it'
Jack says, 'good!'
Jack whisper to Miranda, 'cool!'
```

Team-teaching Lady creates a fun, almost conspiratorial bond between Miranda and Jack.

An interesting problem which arises when trying to teach programming is how to tell people what to type literally and what to fill in with your own content. This is similar to the classic slapstick routine in which the court clerk says, "Repeat after me. I, state your name." The witness replies, "I, state your name." The MOOSE Crossing help system puts in angle brackets things that should be filled in. Jack invents his own solution to this problem by putting words to type literally in capitals and things to fill in in small letters.

```
Lady says, 'So now what do I do?'
Jack says, 'now type DESCRIBE GEM AS "whatever you want!"'
Jack says, 'you do not need to caps it.'
[Lady types: "describe gem as a beutiful white crystal, gleaming in the
   light".  She next looks at the gem to see that it worked.]
Jack whispers to Miranda, 'and we can introduce properities to her by
   making it change color when someone types CHANGE GEM'
Lady says, 'ok, I have the description set'
Miranda says, 'Now, lets say, you want to make it so that when
   someone types 'touch gem' it sparkles'
Miranda says, 'And when you type 'change gem' it changes color.'
Jack grins
Lady says, 'how do you do that, Miranda?'
Jack says, 'drop the gem so we can see it'
Miranda says, 'ok, first you'd have to make a script on it'
[Lady drops the gem.]
Lady drops gem.
fluffey sniffs gem curiously.
Jack says, 'hmm'
Lady says, 'how do you make a script'
Miranda says, 'A script always begins with the word 'on''
Jack says, 'OK, click on the pencil at the top of the screen and type
   in gem. Then check the script circle and type in TOUCH.'
Jack says, 'then hit OK'
Miranda says, 'Tell us when your done'
```

The system design here is less than perfect—Miranda and Jack can't see Lady's screen. A special "last_commands" feature lets you see what someone else has typed (if they allow you), but can not show you the state of another person's windowing environment. The ideal setting for MOOSE Crossing is a combination of spontaneous use and scheduled use in an organized program. Face to face interaction with kids in the same room is complementary to network interaction with kids in far-off places. A local collaborator would be able to see Lady's screen and help her through the basics of creating a script. Luckily, the problem has limited scope—once someone has learned the basics of how to create a script (all of which Lady will learn in this conversation), then he or she can easily receive help on more-advanced topics from others.

```
Jack whispers to Miranda, 'this is cool, i have never tutored anyone
   before!'
Miranda whispers to Jack, 'I have, sometimes it gets frustrating'
```

[Lady fills in the script editor box with the words "the gem sparkles" and clicks Save. This does not create a script.]

```
Lady says, 'I think I'm done'
Miranda says, 'Ok, now do you have a script bow that says 'gem:touch'
   on the top?'
Jack says, 'do you see a large text window?'
Miranda says, 'A meant a script box'
Jack says, 'what do you see?'
Lady says, 'I see a picture of a moose, a pencil, a envelope, and a
   question mark'
Jack says, 'OK, you are still in the main window.'
Lady says, 'so what do I do'
Miranda says, 'Go to 'windows' at the top of your screen, and select
   'gem:touch' if it's there'
Lady says, 'ok'
Lady says, 'it's not there!'
Miranda says, 'Then you haven't created the script yet!'
Jack says, 'click on the pencil and you will get a little window. it
   says edit code at the top. type gem in the box and click on
   script. then hit OK. it will say there is not scripts by that name
   ob the gem. click on the button that says "CREATE AS A SCRIPT"'
Lady says, 'but it does have that script there!'
Miranda pages Amy with 'Jack and I are teaching Lady to program!
```

[She receives a message back that Amy has been idle for over an hour.]

```
Jack says, 'having two people tutor one person is a good idea,
   because as one person is typing, the other person is giving
   instructions :)'
Miranda pages Austina with 'Jack and I are teaching Lady to program!
```

Miranda is taking on a traditionally adult role in teaching Lady to program, and she's eager and proud to show this off to the two adult authority figures logged on, Austina and me. Austina and I are trying to get some work done on improving the system design, and on this afternoon have limited time to come by and hang out with the kids, but we do each stop by briefly to appreciate the work completed.

```
Lady says, 'I don't get this'
Jack says, 'OK, if it does have a script, you should be in, right?'
Miranda says, 'What did you do?'
Lady says, 'I think so'
Jack says, 'yes, describe every move you made:)'
Miranda says, 'Sounds good to me!'
Lady says, 'I went to the pencil and clicked, then I typed gem, I
   clicked on script and clicked "OK" after that it took me to the
   script I made.'
Lady says, 'what did I do wrong?'
Jack says, 'good, what does the script you made say?'
Jack says, 'it should not say anything yet...'
Miranda says, 'Ohhh, so now go to 'windows' and click on your
   script!'
Lady says, 'it says the gem sparkles'
Jack says, 'delete that, you have to put it in a diffrent way.'
Miranda whispers to Jack, 'Here comes the confusing part!'
```

```
Jack whispers, 'as if that first part was not confusing enough!'
Lady says, 'I did type the first line ON touch this.'
 Lady says, 'Do I need a second line with more commands?'
Miranda says, 'That is telling the computer, that when you type
   'touch gem' it should do the following.'
Miranda says, 'Do you get it?'
Lady says, 'yes.'
Jack says, 'yes, you need a second line. On the second line type
   ANNOUNCE "the gem sparkles" or type ANNOUNCE "whatever you want"'
Miranda says, 'Ok, now we're going to type what you want the gem to
   do'
Miranda says, 'That will tell the computer that after someone types
   'touch gem'  it should announce <whatever>'
```

Miranda has tutored others before and is more skilled than Jack at explaining the reasons for things rather than just telling Lady what to do.

```
Lady says, 'ok I've done that'
Austina pages Miranda, 'great! can I come over for a bit?'
Miranda says, 'OK, now, you can type 'end' to end your script. you're
   done!'
Jack says, 'on the third line, type end and hit OK. lets test it out,
   but add more lines to the script to make it better.'
[Lady recompiles the script:
  on touch this
    announce the gem sparkles
  end
It compiles successfully.]
Miranda pages Austina, 'Come on over!'
Jack says, 'er...add more lines to the script later. for now type
   end.'
Austina teleports in.
fluffey sniffs Austina curiously.
Fluffy arrives, following Austina.
fluffey sniffs Fluffy curiously.
[Somewhat confusingly, Austina has a dog named Fluffy and Jack has a dog named fluffey!]
Lady says, 'yes, it is saved. Let's try it out now.'
Jack says, 'hi'
[Jack types 'touch gem'.]
the gem sparkles
[Miranda types 'touch gem'.]
the gem sparkles
Miranda says, 'Lady, did you type 'touch gem'?'
Jack says, 'i did!'
Miranda says, 'Hi!'
Jack grins
Lady says, 'Thanks, Jack and Miranda! I have made my first script.'
Austina applauds loudly!!!
Lady says, 'Yes I did'
[Austina types 'touch gem'.]
the gem sparkles
Austina says, 'yay!'
Lady says, 'now Gem works!'
```

Lady gets immediate positive feedback on her project from two peers and one adult. Her teachers are also her audience.

```
Miranda says, 'You could always add something like telling the person
   something.'
Jack says, 'but we are not done yet... > :)'
Miranda says, 'To do that you could type in 'tell context <whatever
   you want>''
Lady says, 'ok.'
Jack says, 'that will tell only the person who touched it somthing,
   instead of the whole room.'
```

[Lady adds a line to the script:

```
  on touch this
    announce the gem sparkles
    tell context "I am Gem, a smart stone. I am shinny and colorful"
  end
```

The script compiles successfully.]

[Lady types 'touch gem'.]

[Everyone sees:] the gem sparkles

[Lady sees:] I am Gem, a smart stone. I am shinny and colorful

```
Amy pages Miranda, 'cool!  Good luck!'
Miranda pages Amy, 'want to join us?'
Jack says, 'Lady, go back to the script window. lets add more.'
Amy pages Miranda, 'sure, but just for a minute'
Lady says, 'ok'
Miranda says, 'Cool.'
Amy teleports in.
Fluffy sniffs Amy curiously.
fluffey sniffs Amy curiously.
Pumpernickel arrives, following Amy.
Fluffy sniffs Pumpernickel curiously.
fluffey sniffs Pumpernickel curiously.
Amy says, 'hi there!'
Miranda says, 'Hi,'
Austina says, 'hi!'
Jack says, 'hi!'
Amy says, 'nice lab, Jack. My kinda place!'
Lady says, 'Hi'
Miranda says, 'Look at Lady's first script!'
Jack grins
Amy says, 'hiya Lady! Nice to meet you!'
Miranda says, 'type 'touch gem''
```

[Amy types 'touch gem'.]

[Everyone sees:] the gem sparkles

[Amy sees:] I am Gem, a smart stone. I am shinny and colorful

```
Austina says, 'well, guys, it's been cool...   Congrats on your first
   script Lady!  You're doing great!'
Austina says, 'talk to you all later!'
Austina waves
Jack says, 'bye!'
Amy says, 'very nice!'
Austina goes home.
Fluffy follows after Austina.
Miranda says, 'I'm terribly sorry, I have to go have dinner now,
   bye!'
Amy says, 'see ya!'
Lady says, 'Jack, you wnt to suggest some to add on my script?'
Miranda has disconnected.
```

[Amy looks at the gem.]

```
Amy ooohs at the gem.  "Very pretty"
Jack says, 'say OK...'
Jack says, 'er...oK'
Amy hehs
Jack says, 'help dog2'
Jack says, 'aagggh'
```
[Jack now types 'help dog2']
[Lady now looks at 'help dog2' as well.]
```
Amy says, 'great start, Lady!'
Lady says, 'Thanks Jack. I will check the help tomorrow.'
```
[Jack presumably meant to just look at the second part of the dog tutorial for ideas on how to help Lady, but by accident he said it instead of doing it. Lady interprets this as a suggestion that she should read that help message.]
```
Lady says, 'Bye, everyone, I have to go home now.'
Amy waves
Amy goes home.
Pumpernickel follows after Amy.
Jack says, 'your gem?'
Jack says, 'oh well'
[Lady and Jack disconnect.]
```

The next day, Lady connects again and writes another script all on her own, with no help. A few days later, she returns to write four more. She puts these scripts on her dog Cutey (a project she selected for herself, rather than one suggested by her friends.) Here are her scripts:

```
on pet this
  Announce "Cutey rolls over and wags her tail."
end


on feed this
  Announce "Cutey eats the food hungrily, and barks for a thank you."
end


on tickle this
  Announce " Cutey barks, and giggles."
end


on play with this
  Announce" Cutey runs around in circles a few times, while wagging
   her tail."
end


on scratch this
  Announce "Cutey lays down comfortablely enjoying the scratching."
end
```

Lady's learning experience was both community motivated and community supported. She wanted to learn to write a script because Jack had done so. While many people (especially girls) come to think of themselves as the sort who could never do something so high tech, Lady is surrounded by other children writing programs. This helps her to realize that she too can write scripts. Meaningful control over computers is not the privilege of the technical elite—it's within everyone's abilities. Jack and Miranda provided her with project models, role models, technical support, emotional support, and an appreciative audience for her finished work.

Lady's experience would have been very different if, like Uzi, she had logged on later in the evening and found no one else around. Members of the community were an essential part of every aspect of her learning experience.

Real samba schools and The Computer Clubhouse are physical places. People gather there both to work on their projects and to socialize with one another. The architectural space serves as a community center for the members, providing a context for both organized activity and more casual interaction.

# 6. Constructionist Culture

## 6.1 A Felicitous Type of Community

In the previous chapter, I reviewed many ways in which community supports construction activities. Less obvious and equally significant is the fact that the converse is also true: construction activities enhance community.

A community is a group of people brought together for a purpose. The nature of the community is affected by the nature of that purpose, the space (physical or virtual) in which the people interact, and the type of people involved. A particularly felicitous type of community often emerges when people are brought together to construct things. Samba schools in Brazil are an excellent example. Creating a presentation for Carnival brings together a group of people who might not otherwise meet. Creative activity is the motivation for forming the community, and a force which gives shape to the community's activities. In this environment, everyone is constantly learning and helping one another to learn. This is a very different sort of community than, for example, a college fraternity—an organization which often seems designed to prevent people from learning. While this is certainly not true of all fraternities, a substantial number value maximizing alcohol consumption and devalue studying and scholarship. This is a rough caricature, but the underlying point is clear: not all communities have a positive impact on their members.

This chapter argues that communities in which people are making things often take on a special quality. Such communities have what I will call a *constructionist culture*. The chapter reviews a number of issues (for example, allocation of scarce shared resources) which take on particular importance in such communities.

## 6.2 "Television Fans and Participatory Culture"

It's helpful to begin with an example of a kind of community that has little to do with computers: communities of television fans. Many people consider themselves to be fans of television shows; I'm referring in particular to those people who regularly attend conventions and other social gatherings related to these shows. Communities of television fans are a rather surprising example of the positive power of a contructionist culture. When most people think of television fans, they conjure up stereotypes of unattractive and socially inept young men who slavishly memorize obscure facts about *Star Trek* episodes—people who need to "get a life." In *Textual Poachers, Television Fans and Participatory Culture,* Henry Jenkins does a participant-observer ethnography of fan culture, and determines that fans already have a life (Jenkins 1992). Jenkins focuses in particular on fans' creative endeavors. Fans of *Star Trek* and other television shows make videos, publish 'zines, write poetry, and compose folks songs about television shows. Much of the activity at conventions centers on sharing these creative productions.

Underlying Jenkins' research is an exploration of how people make meaning from texts. The fans' creations often make meanings clearly in opposition to the producers' intentions. For example, one genre of fan video implies homosexual relationships between characters like Kirk and Spock: a shot of Kirk looking longingly to his left is followed by a shot of Spock looking longingly to his right, and the sequence is set to the music of a seventies love song. Fans are clearly making their own meanings from the text! Jenkins collects a rich variety of evidence to show that fans are not passive recipients of meanings produced by media conglomerates, but active constructors of personal meanings using popular culture as a source of raw materials. Fundamentally, Jenkins is arguing for a constructivist theory of meaning .

Jenkins uses evidence from fan culture to make the broader point that no viewer is a passive dupe of corporate interests; all viewers make personal meanings from texts. While I believe that this is largely correct, I think it's important not to underestimate the significance of making things. It is through their creative productions that fans are most easily able to construct oppositional readings. While the readers of romance novels studied by Janice Radway (Radway 1984) were clearly able to make their own meanings from the texts (and from the act of reading), they seem to have fewer and less richly nuanced oppositional readings than Jenkins' fans. Reappropriating texts provides rich opportunities for constructing personal meanings.

Much of the richness Jenkins found in fan culture I believe stems from its constructionist nature. Fans not only buy things, they make them. They talk not only about the text, but about their varied re-interpretations of the text, and the processes of their production. These acts of creation restructure not only individuals' relationships to the text, but their relationships to one another. Construction is a community-building activity.

## 6.3 Objects of Construction

The nature of the things being constructed is of course a central factor in the flavor of a constructionist culture. In *The Virtual Community*, Howard Rheingold movingly describes the warm, supportive community that exists in the parenting conference on The Well(Rheingold 1993).[1] The sincerity of the participants' interest in the subject matter is part of what gives the

[1]Building a shared understanding through written or oral conversation is an act of construction in many important senses. This is particularly clear in the case of bulletin board systems like The WELL where the permanent archive of the conversation is clearly something being co-constructed by its participants. However, I do believe that constructing something more substantial than a conversation tends to lead to a more constructionist culture.

conference its special quality. The Grateful Dead conference on The Well is also unusually successful, for the same reason. These topics are personally meaningful to the participants. They are of course not personally meaningful to everyone—not everyone has children, and not everyone likes the Grateful Dead. It's of central importance that participation in those forums is voluntary. The members are a self-selected group of people with a sincere interest in the topic.

Another central factor behind the success of the parenting conference is the fact that the participants have a basic level of shared values and shared understanding about the nature of parenting. There is nothing excessively controversial discussed there. The mood might be quite different if, for example, a vocal minority of the participants insisted on fundamentalist religious approaches to parenting. The USENET newsgroup talk.politics.abortion forms a useful contrast. While the participants in that group certainly care about their subject matter as sincerely and passionately as members of the parenting conference care about the welfare of their children, the controversial nature of the subject matter means that group is far from warm or supportive. In fact, it is dominated by inflammatory rhetoric and *ad hominem* attacks. A minimum level of shared understanding of the nature of the thing being constructed is necessary to create a positive constructionist environment (Bruckman 1996). To draw an analogy to theater, a production company working to prepare a show will have a more positive experience if everyone agrees on the basic form it will take; chaos and conflict are likely to ensue if some members are aiming for a postmodern, highly-stylized Robert Wilson production and others want something more like traditional Gilbert and Sullivan!

While participants must have some degree of shared understanding of the nature of the things being made, the limits must be neither too specific nor too broad. If the objects of construction are too limited, many will fail to be able to express their individuality and find something personally significant within those limits. If the limits are too broad, then members of the community will have too little to say to one another. The design process itself is also more difficult when underconstrained—expert designers know to start with imposing constraints and working within them (Schon 1987).

Any object of construction is of course part of a broader cultural context. In the case of Brazilian samba schools, the joyous nature of Carnival helps make preparing a Carnival performance a community-building act. Any kind of object of construction has connotations which it gets from its role in the broader culture. A Carnival performance has quite different associations than a school assignment. Those associations fundamental shape the learning experience of someone participating in the act of construction.

Research on constructionist learning has long aimed for picking objects of construction which have a low threshold and no ceiling—new learners can become involved with limited effort, but as they progress to become experts, they will still be challenged.

| | MORE SUCCESSFUL | LESS SUCCESSFUL |
|---|---|---|
| Personally meaningful | Sanding a just-finished wood project | Sanding an old wood surface to prevent splinters |
| Voluntarily selected | A garden | School assignment about plant biology |
| Not too controversial | The WELL's parenting conference | USENET's talk.politics.abortion |
| Appropriate scope | A LEGO ship | A LEGO pirate ship made from a pirate-ship kit |
| Cultural associations | Brazil's Carnival | School assembly |
| Low threshold | Logo | Assembly language |
| High ceiling | Logo | Hypercard |
| Low risk | Video | Film |
| High reward | Film making | Lanyard weaving |

Table 6.1: More and Less Successful Objects of Construction

Finally, it is also desirable for the act of construction to have low risks and high potential rewards. It was more difficult to learn to make moving images when each attempt required spending large amounts of money on expensive film stock. Relatively inexpensive and reusable video tapes make moving images a more fertile medium for construction. Lowering expense lowers one kind of risk. Another sort of risk is emotional. Freedom from being judged and evaluated lowers an individual's emotional risk in participating. Formal evaluation does supply one potential source of positive reward. However, the sincere appreciation of a community of one's peers also supplies a strong positive reward with a generally lessened negative risk (unless one's peers are inclined to be scornful).

In summary, objects of construction should ideally be/have:

- personally meaningful,
- voluntarily selected,
- not too controversial,
- neither too limited nor too broad in scope,
- positive cultural associations,
- low threshold,
- high ceiling,
- low risk, and

- high reward.

These factors affect both the experiences of individuals and the patterns of group interaction. Table 6.1 gives examples of a variety of objects of construction which typically meet and fail each of these criteria.

In designing MOOSE Crossing, I tried to meet each of these criteria. Reviewing each in order:

Participation in MOOSE Crossing is generally voluntary (with the exception of some classroom use). Once on MOOSE Crossing, members are free to construct things or not as they wish. When one teacher gave his class assignments of things to do on MOOSE Crossing, a conversation arose on the moose-teachers mailing list. The teachers discussed the educational philosophy of MOOSE Crossing, and concluded that such assignments are not a good idea. To date, every member who has participated for more than a few minutes has chosen to construct new objects—they see others making objects and enjoying making them, and want to make something of their own.

```
CODE OF CONDUCT
To be a member of MOOSE Crossing, you are expected to be a good
citizen.  Most of the rules are just like rules for how to behave
in the real world:

* Don't do anything you wouldn't do at recess at school.
* Do unto others as you would have them do unto you.
* Help others whenever you can.

If you think maybe you shouldn't do something, you probably
shouldn't.  Be nice to other kids.  And help them out.  There's
lots of stuff to learn on MOOSE Crossing.  People will help you
get started, and then you can help the next new person.

These rules you might not have heard before.  They're important:

* Don't tell anyone your home address or phone number.
* Don't meet anyone face to face.

Most people are nice, but there are some people out there who want
to hurt kids.  No matter how nice someone seems or how old they
are, don't tell them your home address or phone number or agree to
meet with them in person.  If you really really want to meet
someone, ask your parents to help you set up a safe meeting in a
public place, and have your parents come with you.

I PLEDGE:
I won't do anything I wouldn't do at recess at school.
I'll treat others the way I want them to treat me.
I will help others whenever I can.
I won't tell anyone my home address or phone number.
I won't meet anyone face to face.
```

Table 6.2: The MOOSE Crossing Code of Conduct (Kid Version)

MOOSE Crossing's code of conduct helps to keep the objects of construction not too controversial. All members agree to a code of conduct when they join. Table 6.2 shows the code of conduct agreed to by kids, and Table 6.3 shows the additional things that rangers (participants 14 and older) agree to. Based on this code of conduct, kids were able to judge for themselves that making guns and bombs would not be appropriate, but making nerf guns and fireworks is fine. These restrictions are perceived as fair because they are a prior condition of participation. The code of conduct helps to maintain harmony within the community.

```
This code of conduct applies even more strongly to adults.  It's
important that you set a positive example for the kids.

RANGER DUTIES
*  Be a positive role model.
*  Be patient and helpful to all kids.
*  Help kids to solve their own social problems by:
   -- Listening to everyone's side of the story,
   -- Asking kids to evaluate their own behavior,
   -- Asking kids to see things from other people's point of view,
   -- Reminding kids of the MOOSE Crossing Code of Conduct, and
   -- Leading a thoughtful discussion.
*  Help kids to solve their own technical problems by:
   -- Explaining general principles,
   -- Pointing out good examples and relevant documentation,
   -- Giving direct answers where appropriate, but
   -- Making sure not to do the work for them.
*  Read the mailing lists *news, *social-issues, and *Rangers
regularly.

RANGER AGREEMENT
I will set a positive example at all times.  I will abide by the
MOOSE Crossing Code of Conduct.  My conduct needs to be better
than acceptable--it needs to be exemplary.  I am a role model for
the members.

I will be patient and helpful.  I will help kids to solve their
own social and technical problems, not solve them for them.

I understand the importance of kids not giving out their home
address or phone number, or arranging to meet people face to face.
To set a good example, I will not give out my home phone number or
address either.  I will not arrange to meet kids I meet on MOOSE
Crossing face to face.  If I see a child violating these rules, I
will explain to them why they should not do so, and notify the
MOOSE Crossing janitors.

I will read *news, *social, and *Rangers on MOOSE Crossing
regularly.
```

Table 6.3: Ranger Addition to the Code of Conduct

The scope of projects possible for MOOSE Crossing is somewhat limited by the code of conduct, but primarily by the affordances of the technology developed. Most unusual about that scope is its limitation to text. Children trying MOOSE Crossing for the first time are usually initially surprised to find no pictures. In a video interview, Mouse (girl, age 9) comments:

> Amy: How is this different from other things you did on the computer before you came here?
>
> Mouse: Well, there are no pictures.
>
> Amy: Is that bad?
>
> Mouse: No, it's good.
>
> Amy: Why is that?
>
> Mouse: Because then you can imagine things. It's better to imagine things than see them on the screen. What if you don't like the picture that they made. And then you write them a letter *(raising her voice and one arm)* "I hate that picture cause I don't see it that way." And so you can see it your own way.

Not everyone finds text an appealing medium. However, for those who do, creating worlds out of text and communicating with people in text opens up new imaginative possibilities, and helps create a meaningful context to develop creative writing skills. The scope of projects possible in this environment is broad enough that people can find a personally meaningful topic, and narrow enough that participants can have a meaningful dialog about their creations.

The cultural associations of MOOSE Crossing are particularly positive: kids associate it with the world of games and free play. Although they are engaging in serious intellectual activity, it is perceived as more play than work. Children's appropriations of elements of popular culture in their creations also bring to the environment the positive associations of those elements.

The threshold for participation in MOOSE Crossing is quite low. I originally expected the activity to appeal primarily to kids 10 and older, and to exceptional nine-year-olds. In practice, several kids as young as seven have been able to participate in a meaningful fashion. Children just learning to read have found MOOSE Crossing to be a fun way to practice reading. The activity also has a relatively high ceiling. Two thirteen-year-olds participating (one boy and one girl) have become accomplished programmers, making elaborate creations enjoyed by the community as a whole, and mastering advanced concepts.

The risks of participation in MOOSE Crossing are low. For Cindy (girl, age 11), it's important that it's not being graded:

Amy: How is writing here different from writing in school?

Cindy: Usually in school you have to do a certain thing. Here you can write whatever you want. And also this computer language or whatever you call it, here it's like "context" and "announce_all_but", and, well, it's like... "fork"...

Amy: How is writing a description different from writing in school?

Cindy: You can make it be whatever you want. It's describing whatever you want. And it's not being graded, which is good!

Cindy is not confident about her school abilities, but she enjoys trying new things and challenging herself on MOOSE Crossing because she feels safe there. There is little risk of failure. Everyone has questions and seeks help from others on their projects when they need it. Social support and freedom from evaluation reduce the risk of trying something new.

Finally, significant rewards follow the completion of even the simplest project. On finishing a new object, children almost always immediately show their creation off to others. The first creations of new members are received with particular enthusiasm, even if they are extremely simple. More elaborate creations become a basis for others' social activity, providing ongoing positive feedback to the designer. The large amount of positive feedback typically received by children is one of MOOSE Crossing's strongest features.

## 6.4 Worlds Made by Their Inhabitants

Not all MUDs are constructionist learning cultures. Far from it. The first MUDs (starting in 1979) were violent hack-n-slash games where participants compete to see who can kill the most monsters and amass the most treasure. Those who win the game become wizards, and only then (if ever), after hundreds of hours of playing time, are they granted the privilege to build a small portion of the world. At the time of this writing, the majority of MUDs still take this form.

The strongest sense of community is adventure-game MUDs is often among the wizards. Once one has won the game and built a castle, there's not much left to do. However, by this time, the participant has invested huge amounts of time in reaching wizard status and in the process has made friends and gained status within the community. Wizards hang around and talk with one another. One veteran MUDder interviewed by Sherry Turkle and myself over the summer of 1992 reported that his desire to talk to the wizards and join that community was his central motivation for trying to win the game (Turkle 1995).

In 1989, a graduate student at Carnegie Mellon University named James Aspnes created a new kind of MUD with no monster or magic swords. He called it “TinyMUD” because the set of available commands was much smaller than the complex variety of combat-oriented commands in other MUDs. Instead, he built in a programming language to allow users to extend the virtual world. People’s main activity went from trying to conquer the virtual world to trying to construct it collaboratively. This led to a much more egalitarian society. In an electronic mail conversation in 1992, I asked Aspnes if that had been part of his goal in creating TinyMUD. He replied:

> You raise an interesting question about the ideals of the TinyMUD community coming from the few founding members. Most adventure-style games and earlier MUDs had some sort of scoring system which translated into rank and often special privileges; I didn't want such a system not because of any strong egalitarian ideals (although I think that there are good egalitarian arguments against it) but because I wanted the game to be open-ended, and any scoring system would have the problem that eventually each player would hit the maximum rank or level of advancement and have to either abandon the game as finished or come up with new reasons to play it. This approach attracted people who liked everybody being equal and drove away people who didn't like a game where you didn't score points and beat out other players (I did put in a "score" command early on since almost everybody tried it, but most players soon realized that it was a joke). I think that this effect created a kind of natural selection which eventually led to the current egalitarian ideals. I like the egalitarianism, but it wasn't my original goal (Aspnes 1992; Bruckman 1992).

One of Aspnes’ primary motivations was to fight “bored wizard syndrome,” the problem of people who have invested so much time in a community later finding themselves with nothing to do because they had “won.” The increased egalitarianism of the new type of MUD was a somewhat unexpected bonus. Permitting and encouraging construction activities created a new kind of community.

There are hundreds of communities descended from TinyMUD where making things (rather than killing them) is the chief activity. Collectively, they are often referred to as “social MUDs.” It’s a remarkable fact that at any given moment there are thousands of people doing creative writing and computer programming in their spare time for fun in these environments. This is what initially interested me in exploring the educational potential of MUDs.

Allowing users to build the world of course has its risks. A world designed by its inhabitants will inevitably have less coherence and more uneven quality than one designed by a single experienced designer. Someone once called MediaMOO “a multicultural mess”—I was never so flattered! While it’s true that the parts of the virtual world don’t form a complete whole, they reflect the rich diversity of the community’s members. A team of Peter Anders’ architecture students at New Jersey Institute of Technology (NJIT) studying the structure of existing virtual worlds immediately assumed that MediaMOO’s more structured Curtis Common area must have been designed first, and loosening control led to disorganization as the space grew. In fact, the opposite is true. MediaMOO was designed to maximize opportunities for individual construction. Much later, a group of regular members desiring more coherence collaborated to add the more structured Curtis Common complex.

There are clearly some situations in which greater coherence is desired, and others in which diversity can be cultivated and enjoyed. Division of the world into public space (with higher coherence and minimum standards) and private space (with greater diversity and opportunity for personalization) makes it possible to resolve this dilemma.

Encouraging users to build the virtual world also helps a community to adapt over time to the changing needs of its members. Themes and patterns emerge and evolve as the user community grows and changes. Users also have a greater sense of ownership in a community which they have helped build. These factors help to create a stronger community.

## 6.5 Sharing Scarce Resources

TinyMUD didn’t stay tiny very long. People built too much. Undergraduates made detailed replicas of their universities. A replica of MIT included much of The Media Lab. In the lab’s cube auditorium was a replica of the set of a play being performed there at that time. A negative review of the play was posted on the wall. Replicas of other universities were equally detailed—room after room went on without end. Eventually, the database got too big for the computer it was running on, and TinyMUD was shut down. Virtual worlds are limited not by availability of land, but by the disk space and processor power of the computers they run on.

Deciding how to allocate scarce shared resources is a key issue that both solidifies and fragments communities, both virtual and real. In any situation in which resources must be shared, some social or political process must evolve to manage those resources. The necessity of sharing resources—whether they be land, fresh air, disk space, or attention of the group’s members—is a hallmark of what makes a community.

My parents live on a street called The Circle, right in the center of a medium-sized town. A few years ago the town wanted to add more parking to increase business for merchants on Main Street. A plan was put forward to seize a number of houses on The Circle by eminent domain to make way for the parking lot. The new lot would increase traffic on all of The Circle and make the neighborhood less attractive. Residents of The Circle formed The Circle Association to protest the plan. They had emergency meetings at people's homes, and recruited local politicians and eminent members of the community to oppose the plan. They wrote letters and spoke with reporters from the local paper. They appealed to other members of the town to consider what kind of a town they really want anyway—is bigger necessarily better? The campaign was ultimately successful—plans for the parking lot were abandoned. In the process, my parents got to know their neighbors in a way they never had before. Years later, members of The Circle Association still meet not only to keep up on the affairs of the town, but also to socialize with one another. A crisis concerning the allocation of scarce resources (land, parking spaces), sparked reflection on the nature of the community (what kind of a town do we want anyway?) and the formation of a strong sub-community.

In the original TinyMUD, anyone could build as much as they liked. This led to overly rapid growth and waste, so a money system was added to control it: people wandering around the world find pennies at random intervals, and a certain number of cents was charged for each thing built. This system too proved unwieldy. Next they added a "builder bit": to be allowed to build, you needed to be granted that permission by the MUD's administrators. This still failed to provide adequate control and explosive growth continued. Many communities descended from TinyMUD switched from pennies to a quota system: each individual gets an initial quota for the number of objects he or she can build. When you have used up your quota, you must request more from the system administrators. While pennies could be accumulated by shear persistence, quota required permission of an administrator. Quota-based systems reintroduced an element of centralized control.

Communities need mechanisms to determine not only how much individuals can build, but where they may build. Most MUDs divide the world into public and private space. In certain special places, sometimes called "residence halls," anyone may build a private home. Off of your private home you may build anything (within the limits of your building quota). To build something in a public place requires permission of the centralized authority.

As communities grow very large, requiring the administrators to judge each request for quota or building in public space can become time consuming and also divisive. To counter these problems, LambdaMOO formed a committee of members called the Architecture Review Board (ARB) to make such decisions. Members of the ARB were initially appointed by the administrators but now are elected.

Conflicts and debates over resource allocation are inevitable in any community with shared resources. They are obviously not always either community-building experiences or learning opportunities for community members—but they can be. How such resource management problems are handled has a strong impact on the flavor of a constructionist culture.

An emerging technical shift promises to change these social dynamics substantially. All early MUD software operated on a centralized, single-server model. All data about the world is stored on one computer, and the controller of that computer ultimately must make resource allocation decisions. An alternative model is to have many interconnected servers. Centralized servers often out grow the memory and processing power of the computer they run on. Much of the current interest in distributed servers is rooted in concern about scalability. A community can grow many orders of magnitude bigger if it can be served off of multiple computers. Whenever the community needs room to grow, another computer can be added. An interesting side effect of this technical change is that it could potentially democratize the management of resource allocation (and many other features of communities as well.) If my part of the world runs on my computer and yours on your computer, then the scope of what I can construct is limited by my financial resources (how big a computer I can buy), rather than by the whim of the administrator of a centralized server. In the future, it may be possible for everyone to control his or her own little corner of cyberspace.

## 6.6 Believing in Users

Many MUDs design spaces for users, rather than letting users build spaces for themselves. In July 1996 I met with researchers from SRI designing an online teacher professional development center in a MUD called Tapped In (Schlager and Schank 1996b). They were neatly organizing virtual offices and meeting rooms. A tremendous amount of effort was being put into anticipating teachers' needs.

I believe Tapped In will be a great resource for teachers. As I argued in Chapter 2, I believe the education community needs to put more effort into supporting teachers rather than replacing them or working around them. Projects like Tapped In are exactly what is needed. However, to my design sensibility, something seemed to be missing from their preliminary design: opportunities for the teachers to extend the virtual world. Instead of trying to anticipate and cater to all of teachers' needs, why not work to give them opportunities to design spaces to meet their own needs?

SRI researcher Mark Schlager replied that he didn't believe that teachers had either the time or the technical expertise to do so. It's clear that for some teachers this is true; for others, it is not. Exactly what proportion have either the time or the confidence necessary I don't know, but I do know that an environment could be designed to accommodate both sorts of people. Teachers with more time could design spaces to meet the needs of those with less time. This is exactly what has happened with MediaMOO's Tuesday Café.

When I founded MediaMOO, I was largely unfamiliar with the field of teacher professional development. MediaMOO members Tari Fanderclai and Greg Siering built a place they called the Netoric (from "network rhetoric") Center. Inside the Netoric Center they built The Tuesday Café (Fanderclai 1996). Every Tuesday night at 8 PM ET they organize a discussion of some aspect of how to use computers to teach writing. Fifteen to sixty teachers usually attend. This appreciative message was posted by a regular Tuesday Café attendee:

```
From:     MikeS
To:       *Anything (#9008)
Subject:  PhD program announcement

MikeS (michael j. salvo IRL) will be attending Texas Tech University
in the fall as a PhD candidate in the English Department's Technical
Communication and Rhetoric concentration.

MediaMOO was my first exposure to the technorhetorical community and
continues to be the most  dynamic and committed group of teachers,
pedagogues, and net-rats i know.  thank you for welcoming me almost
two years ago (!!).  thanks to greg and tari for keeping the tuesday
cafe going, and thanks especially to all the regulars who tolerated
me as a clueless newbie and who have given me hope for the future of
critical educational reform and technorhetoric.  without you all, i
would have left academia -- which *i* consider your greatest gift
(others may not feel the same;-).  i thank you, and i hope our
community lasts a long long time.  thanks, Amy, for keeping this
community going.

mike[2]
```

Notice that Mike refers to "*our* community." It's critical that The Tuesday Café was built by teachers for teachers. It gives them a greater sense of ownership over the space. It also means teachers are taking responsibility for their own professional development, rather than simply following a researcher-designed program for credit. The philosophy of education of The Tuesday Café models the sort of progressive learning environment I would hope that teachers will learn to use with their students.

Also missing from the Tapped In preliminary model is adequate opportunity for social activity. Many regular Tuesday Café attendees return during the rest of the week to play scrabble, order virtual drinks, and just generally spend time with their fellow teachers. Conversations often make an elegant

[2]Posting quoted with permission from the author.

transition from purely social topics to those more clearly work-related, as Rémy Evard noted in his study of The InfoPark, the MUD he used to coordinate activity among his system administrators (Evard 1993). Social activity among peers increases time of participation and reinforces professional development.

Schlager notes that The Tuesday Café is primarily for college and community college writing instructors; elementary school teachers like his targeted Tapped In users have heavier demands on their time and often suffer from greater technophobia. These observations are undoubtedly significant. However, I still believe that Tapped In's users would benefit from greater opportunity and encouragement to build spaces for themselves and their peers.

At the Tenth Computers and Writing Conference, a teacher approached me and proudly told me about a project he was working on. Over the summer he had his graduate students preparing a model of the Pequod in a MUD for his freshman composition students (who would be reading *Moby Dick*) to explore. Won't this be a wonderful educational experience, he asked? Yes, I replied—for the graduate students making the model. Making the model is a much more powerful learning experience than walking through it.

Unfortunately, only a small fraction of MUDs actively encourage users to extend the virtual world, creating new places and objects. Collaborative construction is one of the most interesting and educationally valuable features of these environments. I hope in the future more community founders will trust in their users, encouraging them to make worlds for themselves.

## 6.7 Construction and Community

Diversity is an asset: there are many kind of communities, filling different needs for different sorts of people at different stages of their lives. While I currently regard violent hack-n-slash MUDs somewhat disparagingly, I understand that they serve an important social and recreational (and to a limited extent educational) function for a large number of people. In fact, if I had had access to MUDs when I was 13 or 14, I'm certain I would have loved that sort of environment—I was an avid Dungeons and Dragons player at that age. Different communities serve different needs.

Does this, however, mean that we can make no value judgments among them? I don't believe so—multiculturalism and subjectivity have their limits. Even though as a young teenager I might have enjoyed a violent adventure game environment, I believe that a more constructionist environment would have been better for my social, intellectual, and creative development (and just as much fun). I believe this to be true not just for me at that age but for most people at most ages. This is most certainly a value judgment. The key things I value in a constructionist learning culture are:

- the positive value placed on creative and intellectual activity,
- the ready availability of social support for such activity, and
- a model of learning that is:
    - — self-directed,
    - — self-motivated,
    - — peer-supported, and
    - — life-long.

A constructionist culture can bring these benefits to its members.

# 7. Conclusion: Constructionism and Virtual Communities

This chapter will present a number of open questions for further research, and then summarize the contributions of this thesis.

## 7.1 Open Research Questions

### 7.1.1 The Social Implications of Distributed Systems

MOOSE Crossing runs on a single, centralized server. A distributed model is preferable for both social and technical reasons.

From a technical perspective, a distributed model is necessary for scalability. A centralized server will always be limited by the memory, speed, and disk capacity of the machine it runs on. A system implemented on a distributed architecture can grow many orders of magnitude bigger—perhaps indefinitely. Expansion simply requires additional machines.

This technical change has potentially large social implications. A centralized-server model usually implies centralized control over most policy decisions affecting the community. On MOOSE Crossing and MediaMOO, I make the rules. This includes deciding who gets to be a member of the community, what counts as acceptable conduct, how much each person can build in the virtual world, and even whether the system continues to operate at all. (In other words, it's my sandbox.) MediaMOO briefly experimented with democratic control, but that experiment failed. Part of the reason the experiment failed is the difficulty of avoiding the fundamental fact that the person who controls the hardware and software has ultimate control over the system.[1] If the server is distributed, social control over the virtual world might be more effectively distributed. Everyone could have his or her own small piece of the world, and make the rules for that piece of the world. (Everyone could have his or her own sandbox.)

[1]This is only one of many reasons the experiment failed. Few if any of the active participants had any background in political science, and many mistakes were made. I began the experiment by establishing a voting mechanism to chose an elected council, and left it up to the newly-elected representatives to decide how the council would operate. The new council members chose a consensus-based decision making process, which proved unwieldy and vulnerable to being manipulated by minority interests. Making even minor decisions proved time consuming and difficult, and the elected council members found that the process was taking a great deal of time and emotional energy for little reward. Discussion of even trivial issues became heated as factions began to fight one another for purely political rather than substantive reasons. I tried to make the elected council take on more responsibility and authority than they were willing to accept, and then interfered too much in the process by continuing to participate in ongoing discussions of issues. Early on in the experiment, the not-yet-developed political process was put to a tough test by an accusation of sexual harassment made by a council member against a member of the community. This led to a voluminous, heated, and acrimonious debate which contributed to establishing a hostile atmosphere surrounding the entire political process. Eventually, the council voted to dismantle the experiment and return to autocratic rule.

Distributed architectures do not necessarily imply distributed control over decision making. For example, Fujitsu's WorldsAway graphical world is implemented on a distributed software base for reasons of scalability, but Fujitsu retains complete control over all design decisions, just as is usually the case in single-server models. Distributed architectures do, however, make it much easier to establish distributed social control, where it is desired.

If the economic and technical barriers to establishing your own little piece of the virtual world remain relatively high, then the system that emerges will not be radically different from what exists today. Communities will be more interconnected, and will need to negotiate border issues. Gateways between separate servers may be invisible to the user, or they may come with customs stations checking that you are authorized for access, are not carrying any prohibited objects, and agree to abide by the local rules and regulations. However, overall, things will not be very different.

As the barriers to having your own corner of the virtual world drop, the nature of the medium will likely begin to change more radically. What will be the impact of democratization of control? Will many people want to have their own corner of cyberspace? If so, what will they do there? Are realms run by individuals subject to any broader laws? Whose laws apply? Are there any civil rights in cyberspace? The nature of the social changes this technological shift will facilitate is an intriguing question for further research.

### 7.1.2 The Cognitive Implications of Graphical Media

Many people who visit the Epistemology and Learning Group at the Media Lab are initially surprised to see that MOOSE Crossing is a text-based system. Isn't this the *Media* Lab, they wonder? After they've seen the children's projects, they are usually charmed. The children are using words imaginatively and expressively, developing a new understanding of and love for the written word. After a class in California had been using MOOSE Crossing for a few months, I called their teacher on the phone and asked how things were going. Her first comment was that she couldn't believe the improvement in her students' writing. The students are devoting significant energy to writing, and to revising their writing. They really seem to care about the outcome, because they want to show it off to their peers. For this particular application, the text-based medium supports rich learning experiences.

For many other applications, other media types are preferable. For example, members of MediaMOO would be able to communicate more fluently with one another if they could do so by voice. An educational system designed to promote visual expressiveness would of course be better in a graphical medium. A media type should be chosen by analyzing the unique requirements of each design situation.

Unfortunately, many designers chose to use the flashiest medium possible, regardless of the goals of a specific application. The benefits of text need to be explained to people seeing it for the first time; graphics are more immediately accepted. The initial threshold to using graphical media is often lower, but the limit on what you can ultimately accomplish with it is often lower as well. People seeking commercial success with new media are particularly vulnerable to choosing the highest production values possible to maximize immediate appeal, regardless of the real requirements of the situation.

In reality, higher production values are not always more commercially successful. Fujitsu's WorldsAway two-and-a-half-dimensional virtual world has been much more popular and financially successful than its three-dimensional competitors like AlphaWorld and WorldsChat by Worlds, Inc.[2] One likely reason is that WorldsAway has stronger support for human communication. While AlphaWorld may look much fancier, in WorldsAway your avatar has a wider range of body language and emotional states. Since this medium is ultimately about interpersonal communication, WorldsAway has greater appeal to users than its slicker but less expressive competitors.

This is not an argument in favor of text, or against high production values. It's an argument in favor of the appropriate use of media, and against using graphical media solely for surface appeal.

As of 1997, text-based MUDs are used for many virtual world applications that would be better off being graphical. The reasons are largely economic—most text-based MUD servers are given away for free, are easy to run, and require no computing power on the client side. They can be accessed from almost any computer with any kind of net connection. For many applications such as hack-n-slash games, graphical media are preferable, and will likely replace text-based media in the near future. For example, Diablo by Blizzard Entertainment is a networked Dungeons-and-Dragons-like game which has very quickly gained popularity, with over 5000 players participating each night only a few months after its release. Graphical media offer significant advantages for this application, and it's likely that commercial graphical systems will dominate over freeware text-based systems in the near future.

For educational MUDs, text continues to have pedagogical advantages. However, as children's media companies begin to design large-scale online services, it's inevitable for marketing reasons that those services will be graphical.

[2]This information is from an unconfirmed source.

Currently available graphical systems are much less intellectually engaging than text-based ones. The ability of text-based worlds to encourage creative writing is only part of the reason. Text-based worlds allow users to construct new spaces, and program objects with behaviors. Few graphical worlds allow users to build, and none that I know of created to date (March 1997) allow users to program.

While current graphical technology is less intellectually engaging, this does not have to remain the case. This leads to an intriguing set of design questions. The goal of the design of the MOOSE language and MacMOOSE client was to make the text-based medium more intellectually engaging. How could graphical media be enriched in this fashion? What new learning experiences can graphical media support? Given the inevitability of media for children and adults becoming increasingly graphical, this is an important set of research questions to address in the future.

### 7.1.3 Gender, Technology, and Learning Styles

In the spring of 1992, I bumped into Mitchel Resnick (who was at that time still a graduate student, but about to become a faculty member) in the hallway by the Media Lab's back elevator. I had recently given a presentation about my research into social and psychological phenomena in MUDs to the Media Lab's Narrative-Intelligence Reading Group. That original work was done as a term paper for a course on the sociology of science and technology with Sherry Turkle. My main research at that time was still on interactive cinema. In the hallway, Mitch asked me: What do you think of the idea of making a MUD based on the *Babysitters' Club* series of books to encourage girls to be interested in computing? That was the beginning of what would later become the MOOSE Crossing project—the initial motivation was to explore ideas about gender and computing.

*The Babysitters' Club* is a series of books for young adults published by Scholastic Books that are enormously popular with girls. While that theme might help attract girls, it also seemed a bit restrictive to the imaginative possibilities a virtual world might offer. A theme based on the PBS television show *Ghostwriter* was considered, but also dismissed for similar reasons. I wanted something that would appeal to both girls and boys, and would be fairly open-ended, allowing children to construct anything they imagined. MicroMUSE's theme of the city of the future struck me as being a bit too masculine. On the other hand, a *Babysitters' Club* MUD seemed to go too far in the other direction. The project had no name until June 1993, when I sent this email to a few friends and colleagues:

```
Subject: The social construction of MOOSE
Date: Fri, 25 Jun 93 10:13:56 -0400
From: asb@media.mit.edu

I'm getting awfully tired of referring to "the MUD for kids" and "the
scripting language we're designing."  Names are needed.  And
yesterday I had this wonderful/terrible idea: it's a MOO Scripting
Environment, right?  That sounds like MOOSE to me! So the language
would be called MOOSE.  The place would be called "MOOSE Crossing."
It's a crossing of ways for many different sorts of people.

When you connect, you're at the intersection of two roads.  One way
leads to the city.  Another leads to the country.  There's one tree
in the middle of the clearing.  Dangling from the tree is a horn.  If
you blow the horn, the moose will come.  The moose is a program which
tries to be helpful and answer questions.  When it hasn't been
summoned, it wanders around (usually in the forest, behind the
clearing.)  If you climb the tree, you get to a tree house. If you
climb past the tree house up into the clouds, you get to a kind of
fantasy land.  This is just a place where kids can build new areas
based on whatever fantasy themes they like.  Over all, this is just a
core structure for the kids to build off of.

So here's the gender question:
Clearly we wouldn't want to call a MUD designed to encourage girls to
be excited about computers "racetrack," "boxing rink," or "the
tower."  But I also don't want to call it "teddy bear place,"
"unicorn land," or "rainbow home."  I want something gender neutral
leaning a touch towards the feminine, but not corny.  Does "MOOSE
Crossing" fit the bill?

-- Amy

p.s.  Clearly, the moose is a female moose.  (Does that mean no
antlers?)  What is her name?
```

The idea of having a programmatic moose was later abandoned as unnecessary, but otherwise the structure of the virtual world ended up pretty much as described above.

The idea of making MOOSE Crossing a girls-only environment was considered, and rejected for a variety of reasons. I thought it would be more interesting to construct an environment for both girls and boys, and see whether they reacted differently to it. Additionally, although I'm not fully versed in the literature on single-sex education, my inclination is against it. Girls need to learn to function effectively in the real world, and this includes the presence of boys and men in almost all social contexts. For these reasons, MOOSE Crossing was made a co-ed environment, and its agenda of investigating gender and computing was not announced to participants.

As time went on, gender issues faded in prominence on the research agenda. There seemed to be so much to understand about phenomena like community, construction, and learning in the environment. Gender seemed more appropriate for a follow-on study.

Another reason for delaying research on gender on MOOSE Crossing is methodological. An ethnographic methodology uses detailed observation of particular individuals to try to understand broader phenomena. In the case of studying learning, this methodology seemed appropriate. Applying this approach to the study of gender is more problematic. In examining small groups of individuals, how can we understand what factors are attributable to individuals' personalities, and what factors are correlated with gender?

More quantitative approaches are also problematic. It's difficult to develop a rich understanding of human social behavior from such an approach. We have done some statistical analysis of data recorded on MOOSE Crossing. We divided commands issued into the categories of movement, scripting, looking, help, object manipulation, communication, etc. and compared results for boys and girls.[3] There were no statistically significant differences (See Figure 7.1). Overall, girls and boys appear to be using this medium in the same general ways.

Anecdotally, one teacher of a MOOSE Crossing program at a private school in Minnesota reported that the girls were initially much more interested in MOOSE Crossing than the boys. During this time, the children were creating objects and rooms, spending most of their time writing; no one in the class knew how to write programs. Austina Vainius and I visited the class, and showed the students how to write some simple programs. During that session, the boys and girls appeared to have equal interest. The teacher was quite surprised at the difference from previous sessions. (At the time of this writing, it's not yet clear whether the boys' increased interest will persist over time.)

Of the children who are members of MOOSE Crossing, 44% are girls and 56% are boys. This is a slightly higher percentage of female participants than has been reported for the net in general. Reports of the percentage of women online vary. A survey by the Nielsen company reports that 34% of people going online for the first time in 1995 were female; that percentage rose to 42% in 1997 (Bray 1997). (MOOSE members all joined during this time period.)

It's interesting to break these percentages down by where they (or their parents) heard about MOOSE Crossing and where they connect to MOOSE Crossing. This data is summarized in Table 7.1 and Figure 7.2. When MOOSE is presented to children in the context of a school or after-school

[3]MIT student Austina Vainius did this data analysis, working on the project under the auspices of MIT's Undergraduate Research Opportunities Program (UROP).

program, girls and boys participate in roughly equal numbers. Kids who heard about MOOSE from a friend are equally split along gender lines. Likewise, educational researchers who heard about MOOSE Crossing in a professional context bring an almost equal number of girls and boys to the community (slightly more girls). Data from one girls-only after school program is separated out.

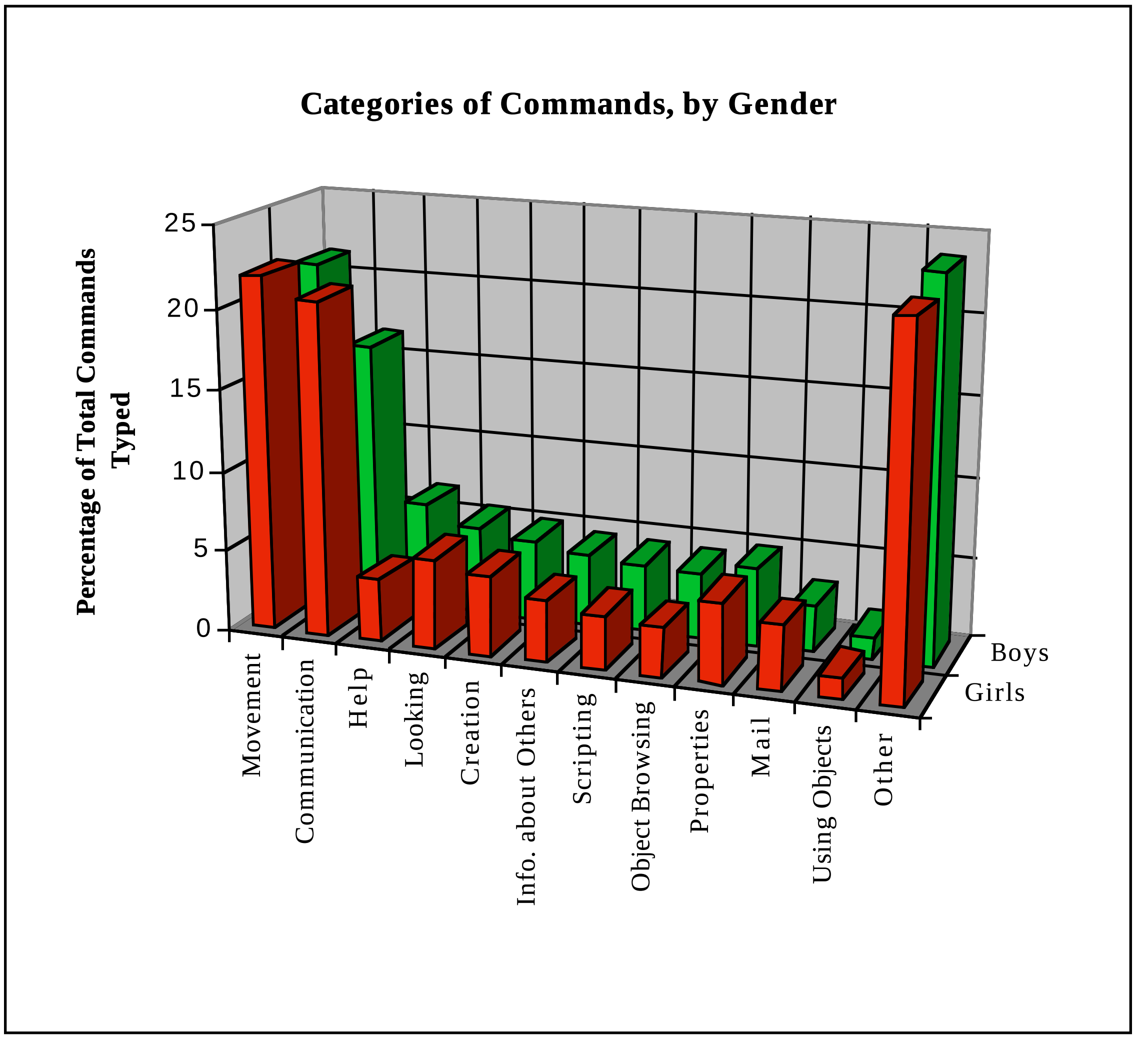


Figure 7.1: Categories of Commands, by Gender

The only gender-unbalanced category is that of kids who heard about MOOSE Crossing via the media (i.e. the popular press, mailing lists, or the web.) More boys are enthusiastic to try MOOSE Crossing and more parents of boys actively seek this educational opportunity for their children than parents of girls. However, usage data indicate that girls tend to like MOOSE Crossing as much as boys. The number of commands typed broken down by gender are presented in Table 7.2. Once they become members, girls and boys participate to a fairly equal degree—the mean number of commands typed is slightly higher for boys, but the median is higher for girls. People expect that boys will like this kind of activity more, but in reality girls and boys like it equally. Our expectations of children's behaviors are more gender-stereotyped than their actual behaviors.

Both ethnographic and statistical methods are limited in their ability to analyze gender-related phenomena. Justine Cassell suggests that a composite methodology where ethnography is used to further explore hypotheses generated by statistical data analysis may be more fruitful. More research is needed to understand gendered phenomena on MOOSE Crossing and in other computational environments.

**How Kids Heard About MOOSE Crossing**
**by Gender**

**Media (i.e. popular press, web, mailing lists)**
Girls: 10 (29%)
Boys: 25 (71%)

**Research Community**
Girls: 15 (54%)
Boys: 13 (46%)

**School**
Girls: 28 (47%)
Boys: 32 (53%)

**Friend**
Girls: 5 (50%)
Boys: 5 (50%)

**Girls-Only After-School Program**
Girls: 9 (100%)
Boys: 0 (0%)

**Other**
Girls: 5 (50%)
Boys: 5 (50%)

Table 7.1: How Kids Heard About MOOSE Crossing, by Gender

**Commands Typed Per Child by Gender**

| | Median | Mean |
|---|---|---|
| Girls: | 354 | 2808 |
| Boys: | 390 | 2139 |
| Both: | 375 | 2440 |

Table 7.2: Commands Typed Per Child, by Gender

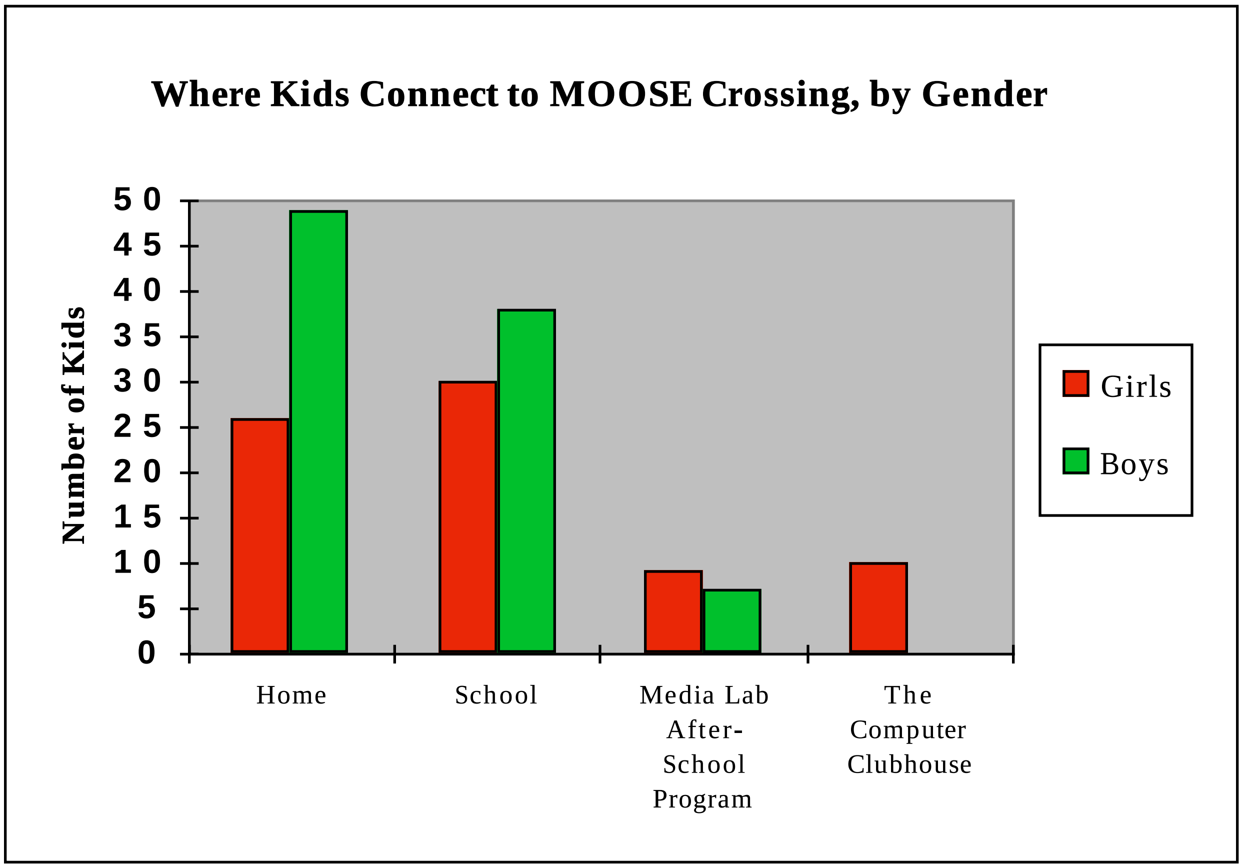


Figure 7.2: Where Kids Connect to MOOSE Crossing, by Gender

## 7.2 Contributions

The primary contributions of this thesis are:

- The creation of a working "technological samba school."
- An articulation of design principles for computer languages (and other technologies) for kids.

- Elucidation of the mutually supportive relationship between construction activities and community, particularly:
    - — An expansion of fundamental understanding of "constructionist" learning, particularly the essential role played by community support for that learning.
    - — An exploration of how constructionism can enhance virtual communities.

This last point is of key importance. The Internet is becoming an increasingly important part of how we work, play, and learn. It remains an open question to what extent communities on the Internet will empower their users. At a colloquium speech at The Media Lab in 1996, David Kurlander, author of Microsoft's Comic Chat client, had a bullet point on one of his slides that said "chat is inane." He smiled and moved on to the next point as if what he had just said was simply a fact of life to accept. Chat is not necessarily inane. Furthermore, interactions among groups of people online can be much more than chat. A constructionist approach to the design of those communities can help to make them more valuable to their members.

A constructionist approach to virtual community design:

- Seeks to maximize each individual's opportunities for creative expression and active participation.
- Starts with the assumption that average people are smarter and more creative than is often assumed, and can achieve great things if given a supportive context to do so. (Applied to software design this is the antithesis of "idiot proofing.")
- Provides well-designed software tools which have a low initial barrier to use, and a high ceiling for what can be accomplished with them.
- Encourages users to be creators of content, maintaining overall quality by enforcing a minimal set of community standards and establishing a distinction between private space and public space.
- Provide opportunities and infrastructure for community support for learning.

In a paper commissioned for the Getty Museum's Art History Information Program, I wrote:

> Cyberspace is not Disneyland. It's not a polished, perfect place built by professional designers for the public to obediently wait on line to passively experience. It's more like a finger-painting party. Everyone is making things, there's paint everywhere, and most work only a parent would love. Here and there, works emerge that most people would agree are achievements of note. The rich variety of work reflects the diversity of participants. And everyone would agree, the creative

> process and the ability for self expression matter more than the product. (Bruckman 1995)

This is a rather different vision than what many designers of corporate web sites have in mind. The Disneyland metaphor dominates the design of many sites whose goal is to display products and services in the best possible light. But the Internet can and will be more than a marketing tool. A constructionist approach can help to make virtual communities empowering for their members, helping groups of people to accomplish what individuals can not accomplish alone.

## Appendix: Five Children's Creations

To give the reader a feeling for what children have been able to accomplish on MOOSE Crossing, this appendix includes all the programs and descriptions of five randomly-selected children. A random-number generator was used to pick them. I decided in advance that I would continue picking children randomly and use the first who fit into each of these categories: a participant in the local after-school program, a home-schooled child, a child who heard about MOOSE Crossing over the net and participates from home, and a child who participates as part of an in-school activity. I also decided that I would keep picking until the full range of ages (7 to 13) was represented. The first four children randomly picked filled most of these categories. I had to pick several more until a child participating from an in-school program was selected.

While this sample was randomly selected, it is unrepresentative in two ways. First, four girls and one boy are included in the sample. The over-all population of MOOSE Crossing is 56% male and 44% female. Second, these children have participated more than is typical. The mean number of commands typed for all children on MOOSE Crossing is 2440 but the mean for this sample is 6994.

Of the children selected, one (Mouse) I would call unusually successful; one (Angela) I would call unusually unsuccessful. Angela participates in school, and has been singled out as having particular difficulty compared to her classmates. The other three children (Goofy, Werdna, and Rowena) are more typical in their level of accomplishment.

Only those objects in existence as of March 1997 are shown. (Mouse in particular has a large number of objects which she recycled when she no longer wanted them.) The children are listed in order of increasing age.

The children's objects are listed in the order in which they were created. This means that the programs they have written are *not* listed in the order in which they were written. For example, Mouse has written a number of programs on her own character object. Your character is the first object you own, so these are listed first. However, these programs are actually relatively advanced programs that she wrote at a much later date. Her first program is on the next object she created, Charlie the caterpillar.

**Goofy**

| | |
|---|---|
| Age: | 7 |
| Gender: | Female |
| Participating from: | Media Lab after-school program; no home access |
| Joined in: | October 1996 |
| Total commands typed: | 153 |

I never anticipated that MOOSE Crossing would be used by kids as young as seven. It's a very different activity for young children who are just learning to read. Goofy is the youngest of four siblings who all participated in the Media Lab's after-school program. Her eldest brother Zoro joined the program first, and the others followed in age order, a few months apart. Zoro had been participating for almost a year by the time Goofy decided to give it a try.

Writing is a painstaking process for Goofy. The long "emote" command in her banana joke script on her monkey took her a few hours of work, spread out over several sessions. She would often pause to ask her older sister (Liza, age 10) how to spell a particular word.

```
*** goofy (parent: MOOSE player class) ****
a fat person with little arms and legs


*** goofy's Room (parent: Generic MOOSE Room) ****
light pink  with blue and  green dots


*** goofy (parent: generic exit) ****
From: Paradise Island
To: goofy's Room


*** out (parent: generic exit) ****
From: goofy's Room
To: Paradise Island


*** a sing tree (parent: generic thing) ****


*** generic monkey (parent: Generic Puppet) ****
he has a red shirt


*** jj (parent: generic monkey by goofy) ****
he has a red shirt and black pants he is a ranbow  monkey

2 scripts on jj:

  on talk
    say knock knock
    say whos there
    say boo
    say boo who
    say "don't cry it's only a  joke"
  end

  on banana
    say "knock knock"
    say "nobody is home"
    emote "reaches into his pocket and pulled out a bunch of bananas"
    emote "reaches into his pocket and pulled out a pietray and then
      said hmmmn how can I do this then he reaches into his pocket
      and pulled's out a portable oven and say's hmmmn how can I do
      this then he unpeels all the bananas smushis the bananas into
      the pietray and puts it in to the oven and wates for it to cook
      then it gos ding rb pushes the button that says cool then the
      oven gos ding rb  pulls the pie out of the oven and smacks itto
      goofys face the end.
  end


*** lemon chime (parent: Generic Puppet) ****
she is brown she loves to eat sweet milk grass she loves to run and
   eat sweet milk grass she understands me and I understand her we
   love each other and she loves to leap over things


******** TOTAL: 2 scripts on 8 objects. ********
```

**Mouse**

| | |
|---|---|
| Age: | 8 to 10 |
| Gender: | Female |
| Participating from: | Media Lab after-school program; additional access from home starting in June 96 |
| Joined in: | November 1995 |
| Total commands typed: | 12972 |

Like Goofy, Mouse wasn’t originally supposed to be participating in MOOSE Crossing—she was, after all, only eight years old. She got involved as a result of a child-care problem. Her ten-year-old sister, Miranda, was one of the first children to try it out. On Miranda’s fourth visit to the Media Lab, their mother Lynn had no place to leave Mouse. Lynn asked if it would be OK if Mouse watched. For three sessions, Mouse watched intently over Miranda’s shoulder, not saying a word. Two things became immediately clear about Mouse: she is extremely shy, and she worships her older sister. Lynn commented that Mouse generally won’t talk to adults at all, unless she knows them extremely well. Every time I asked Mouse if she would like to try MOOSE Crossing too, she responded with a silent shake of the head. At the start of her fourth visit, I asked again. This time, she nodded.

Mouse had clearly been watching carefully. Three and a half minutes after first connecting, she had already found the secret path to the clouds and built a home next to her sister’s room there. She gave it this description:

```
a tiny room with a desk in the center of the room. On the desk there
is a pen and a stuffed teddy bear.  In the corner is a mouse hole. A
mouse sticks its head out.
```

Both her name and room description emphasize a sense of feeling small. After describing her room she joined Miranda and typed ‘say "Hi Miranda".’ Then she didn't say anything, but sat looking at Miranda at the computer next to her, silently. I said out loud to Mouse, "I don't think she's looking at her screen." Miranda heard me and turned to her computer and typed “say hi” back. This conversation ensued:

```
Miranda gives you a hug
Mouse says, "I'm here to hug you!"
Mouse hug Miranda
Mouse hugs Miranda
Mouse says, "I made a mistake"
Miranda smiles
Mouse smiles back
```

Here Mouse types “ask Eddy for joke.” Eddy is one of Miranda’s creations—a giant light bulb that tells light bulb and other jokes.

```
Eddy says, 'What is the best way to catch a squirrel?'
Eddy says, 'Climb up a tree and act like a nut'
Miranda says, "Great joke, right?"
Mouse says, "I guss so."
Mouse hugs Miranda again
Miranda says, "You spelled 'guess' wrong"
Mouse says, "I made a mistake again"
Miranda says, "Everybody makes mistakes."
```

Variations on Miranda’s “on blink this number times” script on Eddy would form the basis for many of Mouse’s projects over the next few months. Initially, she was just following a pattern without fully understanding its component parts. Later on, she would come to understand those parts, and progress to writing more varied and original scripts.

The nine scripts on Mouse’s character object were written at a later date. Her first scripts are those on Charlie the caterpillar.

```
*** Mouse (parent: generic_answering_machine by Miranda) ****
Mouse has Dark brown hair which is worn in a lose ponytail. Her eyes
   are brown and are twinkling happily. She is wearing blue jeans and
   a white, velvet shirt.

9 scripts on Mouse:
```

```
on follow any_person
  set person to any_person
  join person
  fork 500
    join person
    tell me "You are not following anybody anymore."
  endfork
  fork 400
    join person
  endfork
  fork 300
    join person
  endfork
  fork 200
    join person
  endfork
  fork 100
    join person
  endfork
end

set my awake to 1
end

set my awake to 0
end

on float
  tell this "Who would you like to float into?"
  set float to input
  join float
   tell float "Mouse's soul has chosen to float into your body. It
 will follow you where ever you go."
   tell this "Mouse's soul has chosen to float into your body. It
 will follow you where ever you go."
  set my floater to float
end

on exit here by object
  if player is my floater
    fork 5
      this:join player
      tell player "soul is in your body, it followed you"
      tell me "you follow " + player's name
    endfork
  endif
end

on unfloat
  tell my floater "I float out of you"
  tell this "I float out of you"
  set my floater to {}
end

on Easy_Audit
  set Dog to context's owned_objects
  set pigs to {}
  for mouse in dog
```

```
      set rat to nn mouse
      set pigs to (add rat to pigs)
    endfor
    columnize pigs into 3 of width 135
    tell_lines context it
  end


*** Mouse (parent: generic exit) ****
From: Home in the Clouds
To: Main Kitchens


*** Charlie (parent: Generic Following Object) ****
a dark green catorpilor with red bumps on his legs. He has purple
   eyes with bule dots in the middle of them.

2 scripts on Charlie:

  on change this number "times"
    set changed to number
    if changed > 5 times
      emote "bumps turn blue"
    endif
    if changed > 50 times
      emote "bumps turn purplish pink"
    endif
    if changed > 100 times
      emote "bumps turn light green"
    endif
    if changed > 500 times
      emote "bumps start to blink black and brown"
    endif
    if changed > 550 times
      emote "bumps blink purple and pink then turn red"
    endif
  end

  on enter here by object
    emote " wiggles bumps madly and turns a bright yellow."
  end


*** Mouse (parent: generic exit) ****
From: Paradise Island
To: generic_penguin


*** Stacy (parent: Generic Joke-Telling Object) ****
a frendly killer whale. She has Brown eyes and her tail has a rash.

1 script on Stacy:

  on flap this number "times"
    set flapped to number
    if flapped > 5 times
      emote " blinks her eyes happily"
    endif
    if flapped > 50 times
      emote " waves fin as in a way to say hi"
    endif
    if flapped > 100 times
      emote "looks angry"
    endif
    if flapped > 500 times
      emote "wiggles arond madly"
    endif
    if flapped > 550 times
      emote "flips over"
    endif
  end
```

*** R.C. (parent: Generic Puppet) ****
a gray dolphin with a really, really, really shinny tail. He is Stacy's boyfriend.

1 script on R.C.:

```
  On flap this number "times"
    set flapped to number
    set Stacy_here to #xxx member here's contents
    if flapped > 5 times
      if Stacy_here
        emote " jumps over Stacy"
      else
        emote " jumps  up in the air"
      endif
    endif
    if flapped > 20 times
      emote "dances around like a crazy person."
    endif
    if flapped > 50 times
      emote "does the hokey pokey."
    endif
    if flapped > 100 times
      emote " puts his fins on his hips."
    endif
    if flapped > 500 times
      if Stacy_here
        emote "dances on Stacy's head"
      else
        emote " spins like a top"
      endif
    endif
  end
```

*** Newty Cutey (parent: generic_newt by Byron) ****

2 scripts on Newty Cutey:

```
  On crawl this number "times"
    set Crawl to number
    if crawl > 10 times
      emote "slimes you!"
    endif
    if crawl > 50 times
      emote "sticks its toungh out at you."
    endif
    if crawl > 100 times
      emote "turns around dizzlely."
    endif
    if crawl > 500 times
      emote "bites you."
    endif
  end

  On sign_here
  end


*** Mouse's room (parent: Generic MOOSE Room) ****
a chocolate covored room. You lick some of the chocolate off, yummy.
   You lick some more. You can see through the chocolate! You see a
   box. You lick around the box and take it out. It's full of
   chocolate bars! You also see a blue couch and a blue table.


*** Mouse (parent: generic exit) ****
From: Paradise Island
To: Mouse's room


*** out (parent: generic exit) ****
From: Mouse's room
To: Paradise Island


*** generic_bubble_blower (parent: generic thing) ****
A bottle with a labble on it that says "do not touch."  You also see
   a bubble blowing stick taped to the bottle.  You guess it's a
   bubble blower.

2 scripts on generic_bubble_blower:

  on blow bubbles number "times"
    set blow_bubbled to number
    if blow_bubbled > 5 times
      announce_all "bubbles float around in the air."
    endif
    if blow_bubbled > 100 times
      announce_all " bubbles look angry"
    endif
    if blow_bubbled > 500 times
      announce_all " bubbles blow up!!"
    endif
  end
  on blow red bubbles
    announce_all "red bubbles floats in the air"
  end
  on blow blue bubbles
    announce_all "blue bubbles pop on your head"
  end
  on blow green bubbles
    announce_all "green bubbles pop as they float through the air"
  end
  on blow yellow bubbles
    announce_all "yellow bubbles do a silly dance"
  end


*** Dolly (parent: Generic Following Object) ****
 A little doll with a little hat and a little outfit and little
   shoes.

3 scripts on Dolly:

  on drop this
    tell context " I fell down.  Pick me up mommy! "
    pass
    fork (25)
      if this memberof player's contents
        tell context " thank you mommy! Ha!Ha!Ha!  Ha!Ha!Ha! "
      else
        tell context " BAD MOMMY!!! "
      endif
    endfork
  end


  on Somersault this number "times"
    set somersaulted to number
    if somersaulted > 5 times
      emote "Somersaults 5 times and drols happily."
    endif
    if somersaulted > 30 times
      emote "then somersaults 30 times and falls on his head."
    endif
    if somersaulted > 50 times
        emote "then Somersaults 50 times and Tinky tinkles in his
   toilt."
    endif
    if somersaulted > 100 times
      emote " then somersaults 100 times and starts to spin around in
   circles."
    endif
  end


*** Quiet Clearing (parent: Generic Residence Hall) ****
A huge clearing.  On the right there is a freash water pond.  You
   hear birds singing.  It seems like a quiet and cozy place to build
   a home. To build a house, type 'build'


*** Quiet (parent: generic exit) ****
From: Redwood Forest
To: Quiet Clearing


*** out (parent: generic exit) ****
```

```
From: Quiet Clearing
To: Redwood Forest


*** Floppy (parent: generic_greeting_creature by Byron) ****
A cute Puppy with floppy ears. She is black, except her back which is
   white. As you look at her face, You see a gold metal around her
   neck that reads:FLOPPY IS HEREBY NAMED 'OBJECT #xxxx'. Floppy is a
   REALY cute dog.

5 scripts on Floppy:

  on touch this string
    if string is "on the tail"
      say "Hee, hee, hee!"
    endif
    if string is "on the eye"
      say "Ow!"
    endif
    if string is "on the nose"
      say " Don't do that! "
    endif
    if string is "on the head"
      say " Bopy Bop "
    endif
    if string is "on the feet"
      say "watch out I might scratch you"
    endif
    if string is "on the back"
      say " That feels good! "
    endif
    if string is "on the ear"
      say " what did you say? "
    endif
    if string is "on the tummy"
      say "tummy tum tum!"
    endif
    if string is "on the leg"
      say "Hey! You can't touch me there!"
    endif
  end

  on leash_on this
    emote "smiles happily as you put her leash on."
  end

  on tell this string
    if index string "Floppy"
       announce_all_but this "floppy smiles when she hears her name
   being called"
    endif
  end

  On carry this
    tell player "You pick up floppy and pet her on the head."
  end
  on hug this
    tell player "Floppy woofs in delight"
  end


*** Mouse's Plants (parent: Generic MOOSE Room) ****
You are in a Florist. There are plants everywhere!


*** Mr.Planter (parent: generic_creator) ****

2 scripts on Mr.Planter:

  on buy plant
    tel player "What name would you like your plant to have?"
    set plantname to input
    make #xxxx named plantname
    if context not memberof my buylist
      set my buylist to (add context to my buylist)
    endif
  end
  on buy bush
    tel player "What name would you like your bush to have?"
    set bushname to input
    make #xxxx named bushname
    if context not memberof my buylist
      set my buylist to (add context to my buylist)
    endif
  end
  on buy tree
    tel player "What name would you like your tree to have?"
    set treename to input
    make #xxxx named treename
    if context not memberof my buylist
      set my buylist to (add context to my buylist)
    endif
  end
  on buy flower
    tel player "What name would you like your flower to have?"
    set flowername to input
    make #xxxx named flowername
    if context not memberof my buylist
      set my buylist to (add context to my buylist)
    endif
  end

  on enter here by person
    say "Hi," + person's name + " , you are at Mouse's Plants. To buy
   a tree, type buy_tree, to buy a bush, type buy_bush, to buy a
   flower, type buy_flower, and to buy a plant, type buy_plant.  If
   you want to ask a question about a plant, a bush, a flower or a
   tree, page  Mouse with the question or join her and ask her."
  end


*** flu (parent: generic_plant by Miranda) ****
*** proy (parent: generic_flower by Mouse) ****


*** hoyem (parent: generic_bush by Mouse) ****


*** green (parent: generic_tree by Mouse) ****

1 script on green:



*** generic_bush (parent: generic thing) ****

1 script on generic_bush:

  on enter here by object
    announce you see a rabbit scamper away from behind the bush
  end


*** generic_tree (parent: generic thing) ****

1 script on generic_tree:

  on enter here by object
    announce "a bunch of nuts pour out of a hole in the trunk and a
   squirrel poks his head out."
  end


*** generic_flower (parent: generic thing) ****

1 script on generic_flower:

  on enter here by object
    announce a petal drops from the flower
  end


*** Ouija_Board (parent: generic thing) ****
A big board with the alphabet on it and a magnifying piece

1 script on Ouija_Board:

  on use this
    tell context "ask question that initials will answer."
    fork (60)
      set initials to pick my letters
      tell context initials
      set initials2 to pick my letters
      tell context initials2
    endfork
  end


*** Generic_Multiple_personality_puppet (parent: Generic_Assistant by
   Rachael) ****
```

```
4 scripts on Generic_Multiple_personality_puppet:
```

[Note: The scripts on this object were copied by Mouse from a system object, without her really understanding them. Many kids use “generic multiple personality character” to enable them to switch personalities. Mouse decided she wanted her puppet to be able to switch names and descriptions as well. She copied the code from multiple personality character to make multiple personality puppet. I helped her with this. The code isn’t representative of her level of achievement, so I have omitted it here.]

```
*** Mouse's favorite Ice Cream (parent: Generic MOOSE Room) ****
A very big Ice cream store.  There is a big, shiny red counter in the
   center.  Behind the counter, there is a robbot with an apron that
   says Mouse's Helper.  To buy somthing, type buy <flavor>. The Ice
   cream is FREE.  The room is all white except for the floor which
   is covered with a brown carpet.  On the wall, there are pictures
   of famos people eating Ice Cream.  In the back there is a blue
   door that reads: Employees Only.  You like this store a lot. You
   will probably buy some.  The Robot smiles at you warmly and says '
   What flavor would you like.'  You think hard and decide you'll
   get.........


*** Ice_cream_man (parent: generic_creator) ****

1 script on Ice_cream_man:

  on buy string
    tell player "In a cone or a cup?"
    set cko to input
    tell player " You now have " + string + " Ice cream in a " + cko
   + " To lick, type 'lick' now"
    if string memberof my flavors
       tell player " You now have double fudge " + string + " Ice
   cream in a " + cko + "  You get double fudge, because you are a
   friend of Mouse.  To lick, type 'lick' now"
    endif
    set slobber to input
    if slobber is lick
      tell player " You lick your " + string + " Ice cream. "
      tell player "You lick your " + string + " ice cream again."
       tell player "You  lick your ice cream until the Ice cream is
   gone."
    else
      tell player " You tip your ice cream " + cko + "."
    endif
    if context not memberof my buylist
      set my buylist to (add context to my buylist)
    endif
  end


*** Mouse's Nest (parent: Generic MOOSE Room) ****
A tiny ramshakle nest which is falling apart. To fix it up, type
   'describe here as <description'.
```

```
*** Mouse (parent: generic exit) ****
From: Nest Gather Grounds
To: Mouse's Nest


*** down (parent: generic exit) ****
From: Mouse's Nest
To: Nest Gather Grounds


*** Side Hall (parent: Generic Residence Hall) ****
A Very airy, long Hall with a view of a lake. there are lots of open
   windows. On windows, there are lots of plants.

2 scripts on Side Hall:

  on enter here by player
    (pick my open)
    announce_all it
    set my window to it
  end

  on set tempature
    tell player "What would you like the tempature to be? It has to
   be above 40 degrees and under 100 degrees."
    set temp to input
    if temp is "40"
       tell player "The temp is forty, it's mighty cold , you might
   want to turn it up a noch."
      set my temp to 40
    endif
    if temp is "41"
       tell player "The temp is forty one, it is cold in here, you
   might want to turn the heat up more."
      set my temp to 41
    endif
    if temp is "42"
       tell player "The temp is forty two, it is a little cold, you
   should turn it up a little."
      set my temp to 42
    endif
    if temp is "43"
       tell player "The temp is forty three, it's cool in here, you
   should turn it up a little, so people dont shiver."
      set my temp to 43
    endif
  end


*** Mouse's Labratory (parent: Generic MOOSE Room) ****
A very green room with blue carpet. There is a desk with lots of
   bottles full of stuff like, blue liquid and strange possions. Next
   to the desk there is a HUGE window covering tho whole wall! Next
   the window is a tiny table with two books on it and a pad of paper
   and a pen. On the other side of the room there is a bookcase with
   many books about chemisty. Next to the desk there is a little
   stove.  This is a cozy little lab.
```

```
*** Mouse (parent: generic exit) ****
From: Side Hall
To: Mouse's Labratory


*** out (parent: generic exit) ****
From: Mouse's Labratory
To: Side Hall


*** Mouse's bed room (parent: Generic MOOSE Room) ****
You're standing in a small space with a bunk bed on the side. There
   is also a dresser and a wooden desk.  Next to the desk there is a
   bookcase. A closet is next to the bookcase. This is Mouse's bed
   room.


*** Bedroom (parent: generic exit) ****
From: Mouse's Labratory
To: Mouse's bed room


*** Lab (parent: generic exit) ****
From: Mouse's bed room
To: Mouse's Labratory


*** whatsnew (parent: Generic Large-Capacity Mail Recipient) ****
a mailing list with lots of new stuff.


*** LAB (parent: generic exit) ****
From: Mouse's favorite Ice Cream
To: Mouse's Labratory


*** BAL (parent: generic exit) ****
From: Mouse's Plants
To: Mouse's Labratory


*** Down (parent: generic exit) ****
From: Side Hall
To: Main Kitchens


*** up (parent: generic exit) ****
From: <nowhere>
To: Side Hall


*** Plants (parent: generic exit) ****
From: Mouse's Labratory
To: Mouse's Plants


*** Cream (parent: generic exit) ****
```

```
From: Mouse's Plants
To: Mouse's favorite Ice Cream


*** Time Traveler (parent: Generic Viewable Room by Churchill) ****
You see a big Black tube in front of you. On the wall there is a sign
   that looks very old. It reads: 'To travel to a time, Either 1987,
   1984, 1000 bc, 20034 or 90000, just type 'Travel_to <time>'

3 scripts on Time Traveler:

  on Travel_to string
    if string is "1987"
      travel_to 1987
    endif
    if string is "1984"
      travel_to 1984
    endif
    if string is "1000 bc"
      travel_to 1000 bc
    endif
    if strig is "9000"
      travel_to 9000
    endif
  end
  on travel_to 1987
     describe this as "This year was when Mouse was born! You see a
   person holding a baby, The baby's Mouse. You float to a window of
   a house and see the same person, (Mouse's mother) holding Mouse. A
   3 yr. old walks over to Mouse and says 'Ooh! I have a Little
   sister!' That's Miranda. To go Back to 1996, type go_back."
    look
    pick my Says
    tell player it
    tell player "remember, if you want to go type 'go_back'"
  end
  on travel_to 1984
    describe this as "This year was when Miranda was born! You see a
   person holding a baby, The baby's Miranda. You float to a window
   of a house and see the same person, (Mouse's mother) holding
   Miranda. As you look through the window, You see all or Miranda's
   relatives are there. They all never leave her"
    look
    pick my Saysb
    tell player it
     tell player "Remember: if you want to go back to 1996, type
   'go_back'"
  end
  on travel_to 1000 bc
      describe this as "As you look around you see big trees
   everywhere. On rocks carved into chairs, there are people carving
   bows and arows, knitting quilts and molding Bowls. To buy
   something, type buy <thing>. "
    Look
    Tell player "If you want to buy somthing type 'buy <thing>."
    set buy to input
    pass
    tell player "remember, if you want to go type 'go_back'"
  end


  on buy string
    if string is "Bow and arrow"
      Tell player "you walk over to a man carving a bow and arrow and
   ask 'Can you make me one?' the man looks at you oddly, so you
   repet it slowly. He nods his head as if to say 'ok'. He starts
   carving. After awhile he asks you 'what name I carve on bow?'"
      set Bowname to input
      make #xxx named Bowname
    endif
  end
  if string is "Quilt"
     Tell player "you walk over to a women knitting a quilt and ask
   'Can you knit me one?' the women must know a little english
   somehow, because she nods her head as if to say 'ok'. She then
   starts knitting. After a long, long, while, she asks you 'what
   name I knit on quilt?'"
    set quiltname to input
    make #xxxx named quiltname
  endif
  end
  if string is "bowl"
    Tell player "you slowly walk over to a little child molding bowls
   'Can you give me one?' you watch the child walk over to the women
   nitting quilts,(who must be her mother) and says some thing too
   her, she comes back and nods her head. She then picks up a nice
   blue bowl and says, 'what name I carve on bowl on quilt?'"
    set bowlname to input
    make #xxx named bowlname
  endif
  end


  on go_back
    tell player "you go back to 1996"
     describe here as "You see a big Black tube in front of you. On
   the wall there is a sign that looks very old. It reads: 'To travel
   to a time, Either 1987, 1984, 1000 bc, 20034 or 90000, just type
   'Travel_to <time>'"
  end


*** Time (parent: generic exit) ****
From: Mouse's Labratory
To: Time Traveler


*** back (parent: generic exit) ****
From: Time Traveler
To: Mouse's Labratory


*** Generic_Bow_and_arrow (parent: generic thing) ****
```

```
1 script on Generic_Bow_and_arrow:

  on shout this
     Tell player "you shout your bow across the room and it land in
   the wall."
    pick my Shout
    set ho to input
    tell player "you walk over to the wall and" + ho + "it out."
  end


*** Generic_quilt (parent: generic thing) ****

1 script on Generic_quilt:

  on Lay_on this
    If player is #xx
      tell player "You lay softly down on the blacket, (hi amy!).
    endif
  end
  if player #xxx
     tell player "Since you are Miranda, As you lay down a pillow
   appers under your head."
  endif
  end
  if player #xxx
     Tell player "You lay softly on the blacket. As you do a voice
   Comes out of the air and says 'Hi austina!'"
  endif
  end
  if player #xxx
    tell player "HI Mouse! You are a hero! You lay on a gold blacket
   and a pillow slides under your head. Also, a blacket falls down
   over you."
  endif
  end


*** generic_Bowl (parent: generic thing) ****

1 script on generic_Bowl:

  on eat string
    tell player "You eat" + string + "out of your bowl"
  end


*** a sign (parent: Generic Sign by Austina) ****


*** Heavy Cedar Door (parent: generic exit) ****
From: Side Hall
To: Rachael's study


*** Side hall (parent: generic exit) ****
From: Rachael's study
To: Side Hall
```

```
*** Code room (parent: Generic MOOSE Room) ****

2 scripts on Code room:

  on code string
    set answer to ""
    set h to characters string
    for dog in h
      set hog to dog memberof my letters
      set how to item hog of my code
      set answer to answer + how
    endfor
    return answer
  end

  on say string
    set newstr to (code string)
    tell context "You say, " + newstr
     announce_all_but context context's name + " says, '" + newstr +
   "'"
  end


*** Code (parent: generic exit) ****
From: Mouse's bed room
To: Code room


*** back (parent: generic exit) ****
From: Code room
To: Mouse's bed room


*** Hannah/Ileza (parent: generic_conversation by Miranda) ****

1 script on Hannah/Ileza:

  on tell bob string
    pick my Bob
    set bob1 to it
    tell player Bob1
  end


*** Special_Code (parent: generic thing) ****
Code room's code

1 script on Special_Code:

  on read this
    if player is not #xxx
      tell player "You're not Mouse! you just want to read the code!"
    else
    endif
```

```
     tell player "A=@ b=e c=; d=$ e=n f=? g=o h=& i=t j=s k=* l=m m=<
   n=r o=> p=q q=% r=w s=( t=c u=) v=h w=: x=g y=~ z+d  =  .=|!=b
   ?=_"
  end


*** Box (parent: generic container) ****
A box with all of Mouse Generics in it


*** The GreenHouse (parent: Generic MOOSE Room) ****
You stroll down a garden path with cobblestones along the side, you
   enter a huge greenhouse full of exquisite bushes, trees, and
   flowers.  The greenhouse is comfortably warm, and it smells of
   flowers.  There are sprinklers sprinkling cool water on the
   plants.  In this room, you feel at peace with the world.


*** LAB (parent: generic exit) ****
From: The GreenHouse
To: Mouse's Labratory


*** Golumeena (parent: Generic Cliche-Spouting Object) ****
A little round, plump, two inches tall person. She is very cute. But
   has a loud vouce for her size.

2 scripts on Golumeena:

  on poke this number "times
    set poke to number
    if poke > 1
      tell player "Kindly stop poking me"
    endif
    if poke > 5
      tell player "I'll give you one more chance: stop poking me NOW"
    endif
    if poke > 10
      tell player "STOP POKING ME NNNNNNNNOOOOOOOOWWWWWW!!!!"
    endif
    if poke > 20
      emote throw a fit.
      tell player "STOP!!!!"
    endif
    if poke > 50
      tell player "I'll punch you if you don't stop"
    endif
    if poke > 100
      tell player "NOW I'M SO MAD"
      Emote turns red
    endif
    if poke > 500
      tell player "STOP! I CAN'T TAKE IT ANY LONGER!!!!"
      emote crys
    endif
  end
  on enter here by object
    emote quickly jumps out of Mouses pocket
    emote waves
  end


*** Mouse's Room (parent: Generic MOOSE Room) ****
You turn the handle of a red door with a green handle. As you turn
   the handle, it turns into a unique, gold door knob with a lion
   carved into it. You slowly walk into a small room with a nice
   flowery chair in the center. The chair looks as though it is
   magic, it has has a dusty look to it.


*** out (parent: generic exit) ****
From: Mouse's Room
To: turtle


*** Bearuni (parent: Generic_Multiple_personality_puppet by Mouse)
   ****
Bearuni is really Possie, but now Possie thinks she is a white Polar
   Bear!

1 script on Bearuni:

  On Talk_to this
     say "Hi!  " + player's name + " is it?  Well, anyway, can you
   tell me a story? I love storys."
    set story to input
      say "Nice Story! I'm making a book called Possie's book of
   story's. Can you tell me your name so that I can put this story in
   my book?"
    set nme to input
    set Entry to {story, nme }
    set my Stories to (add entry to my Stories)
    say "Thank you SO much! I love when people add a new story to my
   book! When you have time, tell people to add a story to my book.
   Can you tell me an animal? I can become different animals, and I
   need ideas."
    set idea to input
    If idea memberof my Ideas
      say "I already have that name, but thanks any ways."
      return
    endif
    set my Ideas to (add idea to my Ideas)
     say "Thank you! I don't have that animal yet. Do you have any
   ideas on what my name name should be when I become that animal?"
    set name2 to input
    set my Ideas to (add name2 to my Ideas)
    say "That's a lovely name! Can you describe it for me?"
    set descript to input
    set my name to (add descript to my Ideas)
    say "Well, gotta go! See you later!"
  end


*** Possie's Book (parent: generic thing) ****
```

1 script on Possie's Book:

```
  on Scan this
     tell player "You open a small brown book with a gold broken
   clasp. As you look at the contents you see:"
    slice #xxxx.stories 2
    set author to it
    set nut to length author
    for time in 1 to nut
      tell player time + ". " + author[time]
    endfor
  end
```

*** Mouse's Room (parent: Generic MOOSE Room) ****
READ THIS FIRST!!!!!
As you walk into Mouse's room, you see it's not a room at all!  It's a hallway, an endless hallway!  You walk cautionsly onward through the corridor.  As you walk passed the first door, the stench of garbege fills the air.  YUCK!  You quickly read the sign on the first door: STINKY LAND *under construction*  You walk on until you get to the second door. COME IN! reads the purple letters painted onto the door. You turn the door knob. As you look down at your hand you realize there was gum on the door! You peel the gum off into a ball.  Once you do, it flatens out and on it appers the words: Tricked you!  It then falls back onto the door.  You walk on to the next door, dazziled. You look at the sign on the door: COME IN AND HAVE SOME TEA!  Just as you read the last words, you get sucked into the door and apper in a cartoon world! You look down at your hands and realize you're a cartoon drawing too! There is a tree on the right with a little blonde girl on a branch, sleeping. HEY!  You're in the first scene of Alice in Wounderland! The abbreviations for:
The scene where Alice falls down a hole:AH
The scene with the Mad Hatter and the March Hair:MH
The scene where the Queen is playing Croquet:QC
The scene with the Treasure cat:TC
The scene where Alice is on trial:AT
Use the abbreviations to figure out where the exits below lead.

*** Mouse (parent: generic exit) ****
From: Tech City
To: Mouse's Room

*** out (parent: generic exit) ****
From: Mouse's Room
To: Tech City

*** Roo's Room (parent: Generic MOOSE Room) ****
Roo's home to which also belongs to Kanga. It will soon be described better!

*** Roo (parent: generic exit) ****

```
From: The Hundred Acre Woods
To: Roo's Room


*** out (parent: generic exit) ****
From: Roo's Room
To: The Hundred Acre Woods


*** mall (parent: generic exit) ****
From: Mouse's favorite Ice Cream
To: The Grand Moose Mall


*** Lily (parent: generic_female_potato by Miranda) ****


*** Bill (parent: generic_male_potato by Miranda) ****


*** Beely (parent: generic_bee by Miranda) ****


*** little boy potato (parent: generic_male_potato by Miranda) ****
A creamy, buttery, mashed potato.


*** Cheese_Machine (parent: generic_creator) ****
You see a big metal machine built with big metal poles and small red
   buttons. At the end of the machine there is a mini Conveyor belt.
   In the center there is a large button with the Words 'BREAD'
   printed on it. There are 3 differnt levels with little doors on
   the sides of them.

1 script on Cheese_Machine:

  on push red button
     tell player "What kind of bread would you like, Pita Bread,
   hamburger bun, Rye bread, Wheat bread or white Bread? ?"
    set bread to input
    If bread is Pita Bread
      Tell Player "Two little mice pop out of a door. One dressed in
   a pink tu-tu with a Pink bow tied in her hair, The other is
   dressed in a black tux with a black bow-tie at the coller. They
   dance out andpush a small blue button labuled 'claw'"
    endif
  end


*** The Anteater Rentel Shop (parent: Generic MOOSE Room) ****
A big whote room with with and blue tiles on the floor.  There is a
   desk in the back with a sign on it that reads: To Buy a Anteater,
   type buy anteater.


*** Ant (parent: generic exit) ****
From: Mouse's bed room
To: The Anteater Rentel Shop
```

```
*** eater (parent: generic exit) ****
From: The Anteater Rentel Shop
To: Mouse's bed room


*** Anteater1 (parent: generic thing) ****

2 scripts on Anteater1:

  on buy Anteater
    if my Babies is 1
        tell player "You're in luck! There are still babies left!
   Would you like a boy or girl?"
      set gender to input
      if gender is "girl"
         tell player "You pick up a fuzzy anteater with a bow on her
   head. Her long nose sniffs you with her eyes closed.  Suddenly,
   She sneezes all over you!"
      endif
      if gender is "Boy"
        tell player "You pick up a rough anteater with ablue and with
   cap on his head. He puts his hands together and swings them back
   and forth as though swinging a baseball bat.  How Funny!"
      endif
      tell Player "Have good luck with your anteater!  What would you
   like to name it?"
      set name to input
      make #xxxx named name
    endif
    if my babies is 0
       tell player "Tell Mouse, (#xxx) right away that there are no
   more babies, then try again later."
    endif
  end

  on exterminate here
    tell player My name + " sucks up all you ants and get a little
   bigger."
  end


*** AnteaterPen (parent: generic thing) ****
A cardboard box full of anteater babies

2 scripts on AnteaterPen:

  on load this
    set babies to 1
    describe me as "A cardboard box full of anteater babies"
  end

  on unload this
    set babies to 0
    describe me as "an empty cardboard box."
  end
```

```
*** fuzzball (parent: Anteater1 by Mouse) ****


*** JarOfAnts (parent: generic thing) ****

1 script on JarOfAnts:

  On pour this
    tell player "you pour ants all over the room"
     Page here's owner with "Ants are invading one of your rooms!
   Teloport  to  Anteater  rental  shop  and  buy  an  anteater  to
   exterminate your ants!"
  end


*** Kleenex (parent: generic thing) ****
You see a cute, red-brown puppy.  His ears flop around when he walks
   around and his tongue hangs sideways out of his mouth.  His tail
   wags back and forth with exitment. He is a time-telling dog.  To
   see the time, type 'dial Kleenex'.

2 scripts on Kleenex:

  on dial this
     Tell player "You dial the number. Ring! Ring! Goes the phone.
   Soon, you here a voice say: You have reached " + my name + "'s
   Clock.  At the tone, his clock will read," (hhmmss (timestr)) ".
   Beep!!!!! Thank you for calling " + My name + "'s Clock. Call
   again soon!"
    fork 20
       Tell player "Oops! I forgot to tell you, If you want to find
   out the date, type: Dial-date " + My name + "."
    endfork
  end

  on dial_date this
    tell player "You dial:553-3639. Ring! Ring! Goes the phone. Soon,
   you her a voice say: You have reached " + my name + "'s Calendar.
   His calendar says it's" (ddmmyy (Timestr)) ". Thank you for
   calling " + My name + "'s Calender. Call again soon!"
    Fork 20
       tell player "Oops! I forgot to tell you, If you want to find
   out the time, type: Dial " + my name + "."
    endfork
  end


*** Mouse's Useful Pet Shop (parent: Generic MOOSE Room) ****


*** Pet (parent: generic exit) ****
From: Mouse's bed room
To: Mouse's Useful Pet Shop
```

```
*** back (parent: generic exit) ****
From: Mouse's Useful Pet Shop
To: Mouse's bed room


*** Ms.Saleswoman (parent: generic_creator) ****

2 scripts on Ms.Saleswoman:

  on enter here by player
     tell player "Welcome to Mouse's Useful Pet Shop. Here, we sell
   all kinds of useful pets. If you would like to buy: A Time-telling
   dog, type 'buy TimeDog'. Thank you for shopping here at Mouse's
   Useful Pet Shop."
  end

  on buy timeDog
    tell Player "What would you like to name your time-telling dog?"
    set name to input
    make #xxxx named name
  end


*** SUPER_CHEESE =) (parent: generic thing) ****


******** TOTAL: 63 scripts on 93 objects. ********
```

**Angela**

| | |
|---|---|
| Age: | 9 to 10 |
| Gender: | Female |
| Participating from: | In school |
| Joined in: | October 1996 |
| Total commands typed: | 308 |

Angela participates in MOOSE Crossing from school. Her class is scheduled to all log on together for an hour and a half once per week. Many of her classmates chose to log on during recess and other free periods as well. On a visit to Angela’s class, I noted that she seemed less interested in MOOSE Crossing than her classmates. Her teacher commented that Angela often seems uninterested in school activities.

```
*** Angela (parent: MOOSE player class) ****
I am about 5 feet tall and I have brown eyes.I have light brown hair.


*** Angela's Room (parent: Generic MOOSE Room) ****
You see a room with a toy box that has my name on it. Next to the box
   is a red throne with a dog named Faith sitting on it. On one side
   of the room is my bed. Next to my bed is my dresser. My dresser
   has school supplies on it.  Next to the dresser is a blue
   handball.
```

```
*** Angela (parent: generic exit) ****
From: Paradise Island
To: Angela's Room


*** out (parent: generic exit) ****
From: Angela's Room
To: Paradise Island


*** Tiffany (parent: Generic Dog) ****
Tiffany is a grey dog that is about 12 inches tall.She has brown
   eyes.She is very nice.She likes to play fetch.


*** toy box (parent: generic thing) ****


*** toy box (parent: generic thing) ****


*** Angela's Shop (parent: Generic MOOSE Room) ****
You enter Angela's pet shop.In the middle of the shop is the bird
   cage.Look to the right of the bird cage and you see the big
   dogs.Next to that are the bunnies and rabbits.On the left side of
   the bird cage are the puppies.Next to them are the hampsters and
   small pets.To the north side of the bird cage is the counter.


*** Angela (parent: generic exit) ****
From: Crossroads
To: Angela's Shop


*** out (parent: generic exit) ****
From: Angela's Shop
To: Crossroads


******** TOTAL: 0 scripts on 10 objects. ********
```

**Werdna**

| | |
|---|---|
| Age: | 9 to 10 |
| Gender: | Male |
| Participating from: | Home (home schooled) |
| Joined in: | May 1996 |
| Total commands typed: | 9880 |

Werdna is particularly fond of dice games, and has spent a significant portion of his online time teaching other children how to make dice so that they can play dice together.

```
*** Werdna (parent: Generic_Nest_Player by Rachael) ****
a short male elvin wolfrider beside him stands his wolf (nightrunner)
   he is very loyal and will not hurt you

1 script on Werdna:



*** Werdna's room (parent: Generic Atmospheric Room) ****
one door with a lock on it to the north it looks as if it is locked
   and can only be opened from this side so you will have to teleport
   out. blue carpet. its a castle like room with non painted brick
   walls"


*** Werdna (parent: generic exit) ****
From: North Main Street
To: Werdna's room


*** out (parent: generic exit) ****
From: Werdna's room
To: North Main Street


*** Bobo (parent: Generic Dog) ****
You see a large Golden Retriever.

3 scripts on Bobo:

  on tickle Bobo
    say ha ha ha that tickles stop that
    emote rolls on his belly
  end

  on pet
    emote wags tail
  end


*** Werdna's room (parent: Generic MOOSE Room) ****
Green cloud white walls with a roof window to the right one window on
   each wall and a door to the north that has a sign that says
   gambling room for dice games


*** Werdna (parent: generic exit) ****
From: Home in the Clouds
To: Werdna's room


*** out (parent: generic exit) ****
From: Werdna's room
To: Home in the Clouds


*** luckey die (parent: Generic Die) ****
                            |----|
1 script on luckey die:

  on roll die
    (random 5) + 1
    announce my name + " rolls " + it
  end


*** Wackey Toaster (parent: generic thing) ****
A little toaster-person with white hands coming out of the sides of
   its toaster body. its two big eyes and its wide, happy smile make
   you feel like you'd really like some toast.
```

[Note: Werdna's toaster is the result of him following the system's toaster tutorial.]

```
2 scripts on Wackey Toaster:

  on turn knob to string
    set my knob_setting to string
    tell context "You turn the knob on " + my name + " turns the knob
   on "
    + my name + " to " + string + "."
  end

  on make toast
    tell context my name + " gets all excited and puts on a bis simle
   when it realizes that you would like some toast."
     tell context "First, " + my name + " shoves a piece of fresh
   bread in the slot in its belly."
     tell context "Then, " + my name + " takes a deep breath and
   clenches his fists, working hard to make your toast."
    tell context "CLICK!"
    tell context my name + " pops your piece of " + my knob_setting +
   " toast high into the air. it lands straight into your hands."
     announce_all_but context my name + " makes a piece of " + my
   knob_setting + " toast for " + context's name + "."
  end


*** Werdna's castle (parent: Generic MOOSE Room) ****
a huge room with one door to the south


*** castle (parent: generic exit) ****
From: Werdna's room
To: Werdna's castle


*** great hallway (parent: Generic MOOSE Room) ****


*** south (parent: generic exit) ****
From: Werdna's castle
To: great hallway


*** back (parent: generic exit) ****
```

```
From: great hallway
To: Werdna's castle


*** Werdna's gambling room (parent: Generic MOOSE Room) ****
"a big room with tables all over the place. a counter sticking out of
   the wall to the south Werdna's clerk is behind it he says 'wecome
   do you want to play dice. ask somebody to play with you. want some
   beer Ha Ha Ha NOT your to young and i don't have any because this
   is a kids bar or gambling room Ha Ha Ha you kid Ha Ha Ha'


*** gambling (parent: generic exit) ****
From: Werdna's room
To: Werdna's gambling room


*** back (parent: generic exit) ****
From: Werdna's gambling room
To: Werdna's room


*** Werdna's passage (parent: generic exit) ****
From: Werdna's room
To: Werdna's room


*** Werdna's passage (parent: generic exit) ****
From: Werdna's room
To: Werdna's room


*** The arcade (parent: Generic MOOSE Room) ****
"you see many coin games here. (sorry but right know we don't have
   any games programed but when we do we will take this part away.)


*** leave (parent: generic exit) ****
From: The Mall
To: The arcade


*** the mall (parent: generic exit) ****
From: The arcade
To: The Mall


*** ford f - 150 (parent: Generic Vehicle) ****


*** hawk (parent: Generic_Assistant by Rachael) ****
You see a elf as short as Werdna he loves the stars has silver hair
   and blue pants.


*** Werdna's nest (parent: Generic MOOSE Room) ****
```

```
a very nice nest (and pretty big) there are no details but a picture
   to your right the picture is Werdna the elvin wolfrider and his
   wolf (Nightrunner)


*** Werdna (parent: generic exit) ****
From: Nest Gather Grounds
To: Werdna's nest


*** down (parent: generic exit) ****
From: Werdna's nest
To: Nest Gather Grounds


*** fuzzy (parent: Generic Joke-Telling Object) ****
a fuzzy dog like creature but this guy can tell jokes


*** leather shirt and pants (parent: Brown Cloak by Rachael) ****
You see a simple pair of leather pants and a leather shirt which can
   be pulled over ones head. (Quite a feat if I may say so myself.)


*** Werdna's armor and accessories (parent: Generic MOOSE Room) ****
A room full of leather and metal armor he also has wallets and weapon
   holders he will get other things soon.


*** Werdna (parent: generic exit) ****
From: Crossroads
To: Werdna's armor and accessories


*** out (parent: generic exit) ****
From: Werdna's armor and accessories
To: Crossroads


*** if i'm not here read this (parent: generic note) ****
if you want some armor or accessories page me if i'm not connected
   mail me tell me what you want and what you want it made out of and
   any special things you want


*** metal armor (parent: generic thing) ****
bright metal armor that covers the entire body it makes byron look
   very strong and brave and its true he is.

1 script on metal armor:

  on wear this
    this:moveto player
       set  players  description  to  add  my  worn_msg  to  player's
   description
    announce_all_but player player's bame + " " + my oputon_msg
    tell player my pton_msg
  end
```

```
*** golden gold halberd (parent: generic_weapon by Byron) ****


*** fireplace (parent: fireplace by Jack) ****


*** dagger sheath (parent: generic container) ****
a lovely dagger sheath with the name of the person who made it in the
   bottom right hand corner it says Werdna! it is very nice and must
   hold a lovely dagger it is owned by hera she welds it greatly


*** comfy chair (parent: Generic Chair by Austina) ****


*** chainmail (parent: generic thing) ****

2 scripts on chainmail:

  on wear this
    this:moveto player
       set  players  description  to  add  my  worn_msg  to  player's
   description
    announce_all_but player player's name + " " + my oputon_msg
    tell player my puton_msg
  end

  on wear this
    this:moveto player
       set  players  description  to  add  my  worn_msg  to  player's
   description
    announce_all_but player player's name + " " + my oputon_msg
    tell player my puton_msg
  end


*** figaro (parent: Generic Dog) ****
A big male black cat not fat but big with a black and pink nose white
   tummy and white paws

2 scripts on figaro:

  on pet Figaro
    announce_all_but context context's name pets Figaro
    set answer to pick {"purrs and lies down", "runs away from you",
   "walks away from you" }
    tell context my name + " " + answer
  end

  on pick up figaro
    announce_all_but context context's name + " picks up Figaro"
    set answer to pick {"meeows and scratchs + " context's name", "
   jumps out of + " context's names arms", "purrs and rubs up against
   + " context's name" }
    emote answer
  end
```

```
*** anna (parent: Generic Dog) ****
a white and grey female cat pretty small for a grown up cat with a
   white tummy and a white and pink nose with white paws

1 script on anna:

  on pet Anna
    announce_all_but context context's name + " pets anna
    set answer to pick {"purrs and lies down", "runs away from you",
   "walks away from you" }
    emote answer
  end


*** fireball (parent: generic spell_book by Gandalf) ****


*** corgan (parent: generic magic_staff by Gandalf) ****


*** time spender (parent: generic_conversation by Miranda) ****


*** "the mall to the room (parent: Generic MOOSE Room) ****


*** The mall (parent: generic exit) ****
From: The Mall
To: The arcade


******** TOTAL: 13 scripts on 47 objects. ********
```

**Rowena**

| | |
|---|---|
| Age: | 13 |
| Gender: | Female |
| Participating from: | Home (heard about over the net) |
| Joined in: | May 1996 |
| Total commands typed: | 11657 |

Rowena enjoys role playing, and has been a key participant in the medieval role-playing subcommunity started by Rachael.

```
*** Rowena (parent: generic_answering_machine by Miranda) ****
Rowena is a girl with short dark hair.
She is wearing a light forest green dress. It is lightweight and
   seems to 'breathe' in air, making it look cool as a cucumber. It
   looks cheerful and relaxing. Her collar has a white border with
   yellow violets on it. The dress comes down to just below her
   knees, and hangs there. When she walks, it flows around
   majesticly.
```

```
She is wearing a light forest green dress. It is lightweight and
   seems to 'breathe' in air, making it look cool as a cucumber. It
   looks cheerful and relaxing. Her collar has a white border with
   yellow violets on it. The dress comes down to her ankles, and
   hangs there. When she walks, it flows around majesticly.

2 scripts on Rowena:

  if (hug Rowena)
    disp "Rowena smiles and hugs you back!"

  on 'hi Splat
    emote splats Splat on the head.
  end


*** Wheel of Time (parent: generic_book by Rachael) ****
"You see a very old brown leather-bound book. The spine looks as if
   it has been opened and closed so many times that the cover is
   ready to fall off.  On the cover the title " The Wheel of Time" is
   written in flowing gold script.  The pages look yellow with age."


*** storeroom (parent: Generic Viewable Room by Churchill) ****
You are in a dark room lit only by a single candle.  The walls are of
   stone.  Along one wall you see a ladder leading up.  This is a
   viewable room.  type 'view <object>' to see something in more
   detail.

2 scripts on storeroom:

  on grab stone
    if my stone_out is 0
      tell context "You pull the stone out of the wall."
      announce_all_but context context's name + " pulls a loose stone
   out of the wall."
      set my stone_out to 1
      set #xxxx's locked to 0
    else
      tell context "The stone is already out of the wall."
    endif
  end

  on replace stone
    if my stone_out is 1
      tell context "You put the stone back into the hole."
       announce_all_but context context's name + " puts a stone into
   the hole in the wall."
      set my stone_out to 0
      set #xxxx's locked to 1
    else
      tell context "The stone is already in the hole."
    endif
  end
```

Views on storeroom:

up:  Above you you see a square of faint light.  If you climb up to ladder you cluld see it better.

candle:  You see a yellowish wax candle.  It appears to be melting quickly.  The flame dances, throwing strenge shaddows on the wall.

ladder:  You see an old-looking wooden ladder stretching up the wall to a square of light in the celing.  The rungs appear to be simply lashed to the sides, making it look quite unsteady, but hey, you climbed down it to get here.

shadows:  You see strange shadows fliting across the wall.  The look almost as if they are cast by strange beings, not just the candle's dancing flame.

wall:  You see a stone wall with moss growing all over it.  ne of the stones looks loose.

stone:  You see a loose stone.  If you grabed it just right it looks like you could get it out...

*** Mystic (parent: generic thing) ****

2 scripts on Mystic:

```
  on bless object
    announce_all player's name + " chants in Latin."
    emote glows yellow.
     announce_all player's name + " seems to glow white as she lays
   her hands on either side of " + object's name + "'s head."
    announce_all object's name + " is blessed."
  end

  on zap object
    announce_all player's name + " chants in Latin."
    announce_all "A bar of white light shoots out of " + my name + ".
   "
      announce_all "You hear a small rip and " + object's name + "
   suddenly disapears."
    move object to object's home
  end
```

*** Mat (parent: Generic_badger by Zoro) ****
You see a large bager with a beautiful white stripe down his back. He looks like quite a trickster.

*** a hole in the wall (parent: Generic MOOSE Room) ****
The walls are entirely lined with wooden shelves and cabinets.  The shelves have lots of jars and bottles on them as well as some fresh potted herbs and the cabinets are filled with dried herbs. Sitting against the wall there is a large wooden chest.

```
*** hole (parent: generic exit) ****
From: storeroom
To: a hole in the wall

1 script on hole:



*** back (parent: generic exit) ****
From: a hole in the wall
To: storeroom


*** mrMemo (parent: generic_memopad by Miranda) ****


*** Generic Musical Instrument (parent: generic thing) ****
You see a beautiful silver lap harp.  The REAL silver leaf covering
   the harp looks shiny enough to see yourself in, exept for around
   the intricate scroll-work which looks slightly tarnished.

2 scripts on Generic Musical Instrument:

  on sing string
    set my song to string
    tell context my name + " will now sing " + my song + "."
  end

  on play this
    announce_all_but context context's name + " plays " + my song + "
   on " + my name + "."
    tell context "You play " + my song + " on " + my name + "."
    emote rings out with the melody.
  end


*** Bowl of the Winds (parent: generic_Bowl by Mouse) ****
you see a huge bowl. It looks very old. The bottom of the bowl is
   decorated with clouds. When you glance back at the bowl the clouds
   appear to be in a different position.


*** chest (parent: generic container) ****
You see a large wooden chest. It looks very sturdy, though quite old.
   On the front it has a large brass lock.


*** Elcoiwen (parent: generic exit) ****
From: Tech City
To: storeroom


*** up (parent: generic exit) ****
From: storeroom
To: Tech City


*** Flying Toaster (parent: generic thing) ****
```

```
You see one of those silver toasters that's almost round. It has
   wings sticking out of its sides and a flock of toast hovering
   around it.
```

[Note: Rowena’s toaster is the result of her following the system’s toaster tutorial.]

```
2 scripts on Flying Toaster:

  on turn knob to string
    set my knob_setting to string
    tell context "You turn the knob on " + my name + " to " + string
   + "."
    announce_all_but context context's name + " turns the knob on " +
   my name + " to " + string + "."
  end

  on make toast
    tell context "You push down on the little lever on " + my name +
   " and in a minute your " + my knob_setting + " toast pops up."
     announce_all_but context context's name + " pushes down on the
   little lever on " + my name + " and in a minute " + context's pp +
   " " + my knob_setting + " toast pops up."
  end


******** TOTAL: 11 scripts on 15 objects. ********
```

## Bibliography

Appadurai, Arjun (1986). *The Social Life of Things : Commodities in Cultural Perspective*. New York, Cambridge University Press.

Arnold, Michael (1995). “The Semiotics of Logo.” *Educational Computing Research* **12**(3): 205-219.

Aspnes, James (1992). Personal communication.

Baker, Charles L. (1981). “Johnniac Open-Shop System.” History *of Programming Languages* Ed. Richard Wexelblat. New York, Academic Press.

Bartle, Richard (1990). “Interactive Multi-User Computer Games.” MUSE Ltd. *ftp://ftp.lambda.moo.mud.org/pub/MOO/papers/mudreport.txt*

Batson, Trent (1993). “The origins of ENFI.” *Network-Based Classrooms* Eds. Bertram Bruce, Joy Peyton and Trent Batson. New York, Cambridge University Press.

Benedikt, Michael, ed. (1991). *Cyberspace, First Steps*. Cambridge, MA: MIT Press.

Bray, Hiawatha (1997). “Survey: ‘Net usage doubled since late ‘95.” The Boston Globe. Boston, MA, D17.

Brown, Carl (1992). “MicroMUSE History.” *ftp://ftp.musenet.org/micromuse/Muse.History*

Bruce, Bertram, Joy Kreeft Peyton, et al., Eds. (1993). *Network-Based Classrooms*. New York, Cambridge University Press.

Bruckman, Amy (1992). “Identity Workshop: Social and Psychological Phenomena in Text-Based Virtual Reality.” MIT. *ftp://ftp.media.mit.edu/pub/asb/papers/identity-workshop.{ps.Z,rtf.Z}*

Bruckman, Amy (1994). “Programming for Fun: MUDs as a Context for Collaborative Learning.” National Educational Computing Conference, Boston, MA. *ftp://ftp.media.mit.edu/pub/asb/papers/necc94.{ps.Z,rtf.Z,txt}*

Bruckman, Amy (1995). “Cyberspace is Not Disneyland: The Role of the Artist in a Networked World.” Commissioned by the Getty Art History Information Program. *http://www.ahip.getty.edu/cyberpub/bruckman.html*

Bruckman, Amy (1996). “Finding One’s Own Space in Cyberspace.” *Technology Review* **99**(1): 48-54.

*http://web.mit.edu/afs/athena/org/t/techreview/www/articles/jan96/Bruckman.html*

Bruckman, Amy and Mitchel Resnick (1995). "The MediaMOO Project: Constructionism and Professional Community." *Convergence* **1**(1). *http://www.gold.ac.uk/difference/bruckman.html*

Cherny, Lynn (1995). "The MUD Register: Conversational Modes of Action in a Text-Based Virtual Reality." PhD dissertation, Stanford University.

Crowther, Will (1992). Personal communication.

Curtis, Pavel (1993). "LambdaMOO Programmer's Manual." *ftp://ftp.lambda.moo.mud.org/pub/MOO/ProgrammersManual.{ps,dvi,txt}*

Curtis, Pavel (1996). Personal communication.

Dewey, John (1938). *Experience and Education*. New York, Macmillan Publishing Company.

diSessa, Andrea A. and Harold Abelson (1986). "Boxer: A Reconstructible Computational Medium." *Communications of the ACM* **29**(9): 859-868.

Eisenberg, Michael (1995). "Programmable Applications: Interpreter Meets Interface." *SIGCHI Bulletin* **27**(2): 68-83.

Evard, Remy (1993). "Collaborative Networked Communication: MUDs as Systems Tools." Seventh Systems Administration Conference (LISA VII), Monterey, CA, USENIX Association. *http://www.mcs.anl.gov/people/evard/papers/cncmast.html*

Falbel, Aaron (1989). "Friskolen 70: An Ethnographically Informed Inquiry Into the Social Context of Learning." PhD dissertation, Massachusetts Institute of Technology.

Fanderclai, Tari (1996). "Like magic, only real." *Wired Women: Gender and New Realities in Cyberspace* Eds. Lynn Cherny and Elizabeth Weise. Seattle, WA, Seal Press.

Feurzeig, Wally (1984). "The Logo Lineage" *Digital Deli*. Ed. Steve Ditlea. New York, NY, Workman Publishing Company.

Feurzeig, Wally (1996). Personal communication.

Frazier, Frank (1967). "The Logo System: Preliminary Manual." Cambridge, MA, Bolt Beranek and Newman.

Geertz, Clifford (1973). *The Interpretation of Cultures*. New York, Basic Books.

Glusman, Gustavo, E. Mercer, et al. (1996). “Real-time Collaboration On the Internet: BioMOO, the Biologists’ Virtual Meeting Place.” *Internet for the Molecular Biologist*. Eds. S.R. Swindell, R.R. Miller and G.S.A. Myers. Norfolk, UK, Horizon Scientific Press.

Goffman, Erving (1959). *The Presentation of Self in Everyday Life*. New York, Doubleday.

Goodman, Danny (1988). *The Complete HyperCard Handbook*. New York, Bantam Books.

Guzdial, Mark (1994). “Software-Realized Scaffolding to Facilitate Programming for Science Learning.” *Interactive Learning Environments* **4**(1): 1-44.

Harel, Idit (1991). *Children Designers*. Norwood, NJ, Ablex Publishing.

Holdaway, Don (1979). *The Foundations of Literacy*. New York: Ashton Scholastic.

Honey, Margaret, Babette Moeller, Cornelia Brunner, Dorothy Bennett, Peggy Clements, and Jan Hawkins (1991). “Girls and Design: Exploring the Question of the Technological Imagination.” Technical Report No. 17. New York: Bank Street College of Education.

Hooper, Paula (1997a). Personal communication.

Hooper, Paula (1997b). “Their Own Thoughts.” MIT.

Hughes, Billie (1996). Personal communication.

Hughes, Billie and Jim Walters (1995). “Children, MUDs, and Learning.” AERA, San Francisco, CA.
*http://pcacad.pc.maricopa.edu/Pueblo/writings/bib/AERA-paper-1995.html*

Hughes, Billie and Jim Walters (1997). Personal communication.

Jenkins, Henry (1992). *Textual Poachers, Television Fans and Participatory Culture*. New York, Routledge.

Kaehler, Ted (1996). Personal communication.

Kay, Alan (1996). “The Early History of Smalltalk.” *History of Programming Languages-II* . New York, ACM Press.

Keller, Evelyn Fox (1985). *Gender and Science.* New Haven: Yale University Press.

Kleinschmidt, Klaus (1996). “The Internet is a waste of time for schools.” The Boston Globe. Boston, 22.

Klietz, Alan (1992). Personal communication.

Kort, Barry (1997). Personal communication.

Kurtz, Thomas (1981). “BASIC.” *History of Programming Languages* Ed. Richard Wexelblat. New York, Academic Press.

Latour, Bruno and Steve Woolgar (1986). *Laboratory Life: The Social Construction of Scientific Facts.* Princeton, NJ, Princeton University Press.

Laurel, Brenda, Ed. (1990). *The Art of Human-Computer Interface Design*. Reading, MA, Addison-Wesley.

Laurel, Brenda (1991). *Computers as Theatre.* Reading, MA, Addison-Wesley.

Lave, Jean and Etienne Wegner (1991). *Situated Learning: Legitimate Peripheral Participation.* Cambridge, UK, Cambridge University Press.

Leron, Uri (1985). “Logo Today: Vision and Reality.” *The Computing Teacher*.

Levin, James A., Al Rogers, et al. (1989). “Observations on educational electronic networks: The Importance of appropriate activities for learning.” *The Computing Teacher* **16**:

Morningstar, Chip and F. Randall Farmer (1991). “The Lessons of Lucasfilm’s Habitat.” *Cyberspace, First Steps.* Michael Benedikt, editor. Cambridge, MA, MIT Press.

Newman, Denis, Peg Griffin, et al. (1989). *The Construction Zone: Working for Cognitive Change in School.* Cambridge, England, Cambridge University Press.

O’Day, Vicki (1997). Personal communication.

O'Day, Vicki, Daniel Bobrow, et al. (1996). "The Social-Technical Design Circle." CSCW 96, Cambridge, MA, ACM Press.

Oldenburg, Ray (1989). *The Great Good Place.* New York: Paragon House.

Papert, Seymour (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. New York, Basic Books.

Papert, Seymour (1991). "Situating Constructionism." *Constructionism* Eds. Idit Harel and Seymour Papert. Norwood, NJ, Ablex Publishing. 518.

Pea, Roy (1993). "Practices of Distributed Intelligence and Designs for Education." *Distributed Cognitions: Psychological and Educational Considerations* Ed. Gavriel Salomon. Cambridge, England, Cambridge University Press.

Postman, Neil (1985). *Amusing Ourselves to Death, Public Discourse in the Age of Show Business*. New York, Viking.

Putnam, Robert (1995). "Bowling Alone: America's Declining Social Capital." *Journal of Democracy* **6**(1).

Radway, Janice (1984). *Reading the Romance: Women, Patriarchy, and Popular Literature*. Chapel Hill, University of North Carolina Press.

Raymond, Eric (1991). *The New Hackers Dictionary*. Cambridge, MA: MIT Press.

Reid, Elizabeth (1991). "Electropolis: Communication and Community on Internet Relay Chat." Bachelors thesis, University of Melbourne. *ftp://ftp.lambda.moo.mud.org/pub/MOO/papers/electropolis.{ps,txt}*

Resnick, Mitchel (1993). Personal communication.

Resnick, Mitchel (1994). *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. Cambridge, MA, MIT Press.

Resnick, Mitchel, Amy Bruckman, and Fred Martin. "Pianos, Not Stereos: Creating Computational Construction Kits." *Interactions* September/October 1996, 40-50.

Resnick, Mitchel (1997). "StarLogo Reference Manual." *http://www.media.mit.edu/~starlogo/documentation/refman.html*

Resnick, Mitchel, Robert Berg, et al. (1997). “Beyond Black Boxes: Brining Transparency and Aesthetics Back to Scientific Instruments.”

Resnick, Mitchel and Natalie Rusk (1996). “The Computer Clubhouse: Preparing for Life in a Digital World.” *IBM Systems Journal* **35**(3-4): 431-440. *http://el.www.media.mit.edu/groups/el/Papers/mres/Comp_club/Clubhouse.html*

Resnick, Paul and James Miller (1996). “PICS: Internet Access Controls Without Censorship.” *Communications of the ACM* Forthcoming. *http://www.w3.org/pub/WWW/PICS/iacwcv2.htm*

Rheingold, Howard (1993). *The Virtual Community: Homesteading on the Electronic Frontier*. Reading, MA, Addison-Wesley Publishing Company.

Rogoff, Barbara (1994). “Developing Understanding of the Idea of Communities of Learners.” *Mind, Culture, and Activity* **1**(4): 209-229.

Rubin, Andee and Brad Goodman (1991). “TapeMeasure: Video as Data for Statistical Investigations.” Proceedings of American Educational Research Association, Chicago.

Sammet, Jean (1981). “The Early History of COBOL.” *History of Programming Languages* Ed. Richard Wexelblat. New York, Academic Press.

San Jose Mercury News (1996). “Clinton Makes Cyberspace Connection.” San Jose Mercury News. San Jose, CA, 18A.

Scardamalia, Marlene and Carl Bereiter (1994). “Computer Support for Knowledge-Building Communities.” *The Journal of the Learning Sciences* **3**(3): 265-283.

Schlager, Mark and Patricia Schank (1996a). “A Networked Environment for Teacher Professional Development.” SRI. *http://tappedin.sri.com/info/concept.html*

Schlager, Mark and Patricia Schank (1996b). “TAPPED IN: A Multi-User Virtual Environment for Teacher Professional Development and Education Reform.” The Virtual Classroom, Berkeley, CA, *http://www.soe.berkeley.edu/~schank/TappedIn.html*

Schon, Donald A. (1987). *Educating the Reflective Practitioner*. San Francisco, Jossey-Bass.

Shaffer, David Williamson (1996). “Escher’s World: Learning Mathematics through Design in a Digital Studio.” MIT.

Shaw, Alan (1994). “Social Constructionism and the Inner City: Designing Environments for Social Development and Urban Renewal.” PhD dissertation, Massachusetts Institute of Technology.

Skinner, B. F. (1968). *The Technology of Teaching*. New York, Appleton-Century-Crofts.

Songer, Nancy (1996). “Exploring Learning Opportunities in Coordinated Network-Enhanced Classrooms: A case of kids as global scientists.” *The Journal of the Learning Sciences* **5**(4): 297-327.

Stone, Allucquere Rosanne (1991). “Will the Real Body Please Stand Up?: Boundary Stories about Virtual Cultures.” *Cyberspace, First Steps*. Michael Benedikt, editor. Cambridge, MA, MIT Press.

Strohecker, Carol (1991). “Why Knot?” PhD dissertation, Massachusetts Institute of Technology.

Turkle, Sherry (1984). *The Second Self: Computers and the Human Spirit.* New York, Simon and Schuster.

Turkle, Sherry (1986). “Computational Reticence: Why Women Fear the Intimate Machine.” *Technology and Women’s Voices* Ed. Cheris Kramerae. New York, Pergamon Press.

Turkle, Sherry and Seymour Papert (1992). "Epistemological Pluralism and the Revaluation of the Concrete." *Journal of Mathematical Behavior* 11:3-33.

Turkle, Sherry (1995). *Life on the Screen: Identity in the Age of the Internet*. New York, Simon & Schuster.

Van Buren, David, Pavel Curtis, et al. (1994). “The AstroVR Collaboratory.” *Astronomical Data Analysis Software and Systems IV* Eds. R. Hanish and H. Payne. San Francisco, Astronomical Society of the Pacific.

Vygotsky, Lev (1978). *Mind in Society*. Cambridge, MA, Harvard University Press.

Wall, Larry (1996). “Wherefore Art, Thou?” *The Perl Journal* **1**(1): 5 - 7.

Walters, Jim and Billie Hughes (1994). “Camp MariMUSE: Linking Elementary and College Students in Virtual Space.” National Educational Computing Conference, Boston, MA, Weir, Sylvia (1992). “Electronic Communities of Learners: Fact or Fiction.” Cambridge, MA: TERC Communications.

Winner, Langdon (1986). *The Whale and the Reactor*. Chicago: University of Chicago Press.

Winograd, Terry and Fernando Flores (1987). *Understanding Computers and Cognition*. Reading, MA, Addison-Wesley.