

CEL: A Framework for Enabling an Internet Learning Community

A Dissertation

Presented to

The Faculty of the Graduate School of Arts and Sciences

Brandeis University

Department of Computer Science

Jordan B. Pollack, Advisor

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

by

Elizabeth Ida Sklar

May 8, 2000

This dissertation, directed and approved by Elizabeth Ida Sklar's Committee,
has been accepted and approved by the Graduate Faculty of Brandeis
University in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Dean of Arts and Sciences

Dissertation Committee

Professor Jordan B. Pollack, Chair

Professor Richard Alterman

Professor Pattie Maes

Professor James Storer

Copyright © by

Elizabeth Ida Sklar

2000

for my two grandmothers, Mildred and Sadie

*the faces of my yesterday who gave me the strength and the courage to pursue the dreams
of my tomorrow*

and my three children, Suzanne, Jennifer and Alex

the faces of my tomorrow who make all the todays worthwhile

Acknowledgments

Many people have helped me along the journey that has culminated in this thesis. I would like to offer special thanks to the following —

To my advisor, Jordan Pollack, for supplying an endless stream of new ideas, for trusting in me and for being my friend. To my committee, Rick Alterman, Pattie Maes and Jim Storer for their wisdom and advice and for taking an interest in my work. To Tom Banaszewski and Jackie Kagey, for supporting the pilot study.

To the members of my department, with whom I've shared friendship and sometimes authorship over the past 5 years: Paul Buitelaar, Carina Canaan, Paul Darwen, Jeanne DeBaie, Sevan Ficici, Andy Garland, Greg Hornby, Hugues Juillé, Simon Levy, Hod Lipson, Ofer Melnik, Julio Santana, Miguel Schneider, Marc Verhagen, Richard Watson, and especially to Pablo Funes, for many an afternoon hack and for his devotion to Tron, with *besitos*. To the faithful stream of undergraduate programmers who helped me implement the CEL system: John Abercrombie, Robert Gebhardt, Travis Gebhardt, Matthew Hugger, Louis Lapat and Maccabee Levine.

To Myrna Fox, for listening patiently to my daily woes and always giving me sound advice. To my sister, Deborah Sklar, for supplying the beautiful artwork that became the CEL logo. To Ed Rozier, for being an ever-faithful babysitter and father. To my ever-faithful babysitter and friend, Linea Hopwood, who always kept my guys safe and sound.

To my parents, Jay and Ellen Sklar, for teaching me to stand up on my own two feet and for letting go, because you always knew I wanted to do it all by myself; for loving me despite my mistakes and for being proud of me.

To my guys — Suzanne, for your enthusiasm and willingness to test anything and everything; Jennifer, for the best hugs ever and for continually enlightening me by carefully explaining the world to me; and Alex, for quietly growing from an infant into a capable little boy, waiting patiently each day for your frantic mother to get the girls on the bus so we could finally sit down and share our breakfast together.

And to Alan Blair, my best friend and wonder twin, for all your love and support throughout this process, at any time of day or night, through lice and floods and bone infections. You always believed in me, even when no-one else did, even when I lost faith in myself. Thank you, from the bottom of my heart.

Go team!

— *Betsy*.

ABSTRACT

CEL: A Framework for Enabling an Internet Learning Community

A dissertation presented to the Faculty of
the Graduate School of Arts and Sciences of
Brandeis University, Waltham, Massachusetts

by Elizabeth Ida Sklar

With the introduction of personal computers into schools, educational software has infiltrated classrooms. Yet despite all the technology that is currently available, the order of magnitude improvement in student performance which many expected as a result of computer-based interactive learning systems is not evident. There are many reasons for this shortcoming. This thesis discusses these reasons and responds to three specific issues: the practical needs of schools, the curricular needs of teachers, and the overlapping system needs of researchers.

We present a prototype system called CEL (*Community of Evolving Learners*) which provides an environment that is: accessible, because it can be reached by schools with minimal hardware capabilities and no specialized software installation requirements; flexible, because it can host a variety of curricular and experimental activities; and extensible, because it offers a shareable framework to which others can add their own activities.

CEL is an Internet environment in which users engage in multi-player educational games, with each other and/or with software agents. This thesis describes the CEL system, detailing its design and explaining the kinds of activities that can be hosted by CEL and the types of data that can be gathered. Pilot testing that was used to validate the CEL mechanism is outlined. Throughout, we demonstrate CEL as an accessible, flexible and extensible platform capable of supporting many types of curricular activities and research experiments.

Contents

1	Introduction	1
1.1	Contribution	3
1.2	Outline	7
2	Background	9
2.1	What are interactive learning systems?	11
2.1.1	Instructive learning systems	11
2.1.2	Constructive learning systems	12
2.2	How are interactive learning systems evaluated?	13
2.3	Interactive learning studies	16
2.3.1	Experimental research: motivation	17
2.3.2	Experimental research: skill acquisition	21
2.3.3	Field study: KIE	24
2.3.4	Field study: Pueblo	25
2.3.5	Field study: MOOSE Crossing	28
2.3.6	Pilot study: MANIC	30

2.3.7	Pilot study: Counting on Frank	31
2.3.8	Pilot study: Phoenix Quest	32
2.3.9	Pilot study: ScienceSpace	33
2.3.10	Pilot study: Zadarh	34
2.3.11	System design: Belvedere	35
2.3.12	System design: CoVis	36
2.4	Summary	38
3	An Overview of CEL	45
3.1	A brief tour of CEL	45
3.2	The IDsigner	51
3.3	Prototype Activities	53
3.3.1	Keyit	53
3.3.2	Pickey	56
3.3.3	Monkey	58
3.3.4	Automath	60
3.3.5	Loois	62
3.3.6	Tron	64
3.4	Summary	66
4	System Architecture	67
4.1	Server	69
4.2	Messenger	73
4.3	Monitor	73

4.4	Database Manager	74
4.5	Matchmaker	75
4.6	Secret Agent	76
4.7	Player	78
4.7.1	Formative assessment	79
4.7.2	Final design	83
4.8	CEL Message Language	84
4.9	Player States	89
5	Data in CEL	93
5.1	Domain knowledge	94
5.1.1	<i>Words</i> database	96
5.1.2	<i>Arithmetic</i> database	99
5.2	Data products	101
5.2.1	Student model	101
5.2.2	Performance	103
5.2.3	Match Results	106
5.2.4	Survey Results	108
5.2.5	System Logs	109
5.3	Summary	110
6	Pilot Testing	113
6.1	Activity	118
6.2	Interaction	121

6.3	Learning	127
6.4	Interest	129
6.5	Off-line survey	131
6.6	Summary	135
7	Agents as learning partners	137
7.1	Functional description	138
7.2	Playground behavior	141
7.3	Game behavior	143
7.4	Training agents to emulate humans	145
	7.4.1 Architecture	147
	7.4.2 Training	148
	7.4.3 Results	150
7.5	Discussion	154
8	Domain coverage	155
8.1	The domain	157
8.2	Selection algorithm	158
	8.2.1 Merging	161
	8.2.2 Reproduction through sampling	164
8.3	Results	169
	8.3.1 Domain coverage	170
	8.3.2 Feature correlation	172
8.4	Discussion	177

9	Conclusion	179
9.1	Accessibility	179
9.2	Flexibility	180
9.3	Extensibility	181
9.4	Issues in Internet communities	181
9.4.1	Safety and privacy in CEL	182
9.4.2	Identity in CEL	185
9.4.3	Communication in CEL	187
9.5	Future work	188
9.5.1	Visualization	189
9.5.2	Player clustering	189
9.6	Finally	193

List of Figures

2.1	System development cycle	16
3.1	CEL Home Page.	46
3.2	Logging in to CEL.	47
3.3	The CEL menu.	48
3.4	The CEL Playground.	49
3.5	An invitation to play a match.	50
3.6	Site Map.	51
3.7	The IDsigner.	52
3.8	The game of Keyit.	55
3.9	The game of Pickey.	57
3.10	The game of Monkey.	59
3.11	The game of Automath.	61
3.12	The game of Loois.	63
3.13	The game of Tron.	65
4.1	System Architecture.	67

4.2	Sockets.	70
4.3	Server architecture, with overview of clients.	72
4.4	The Monitor.	74
4.5	The Matchmaker	75
4.6	Typical software agent architecture.	77
4.7	Overview of the player.	79
4.8	The CEL Playground, initial version.	80
4.9	The CEL Playground, intermediate version.	82
4.10	Player state diagram.	90
5.1	Sample word with feature vector.	98
5.2	RATE definition for Keyit, Pickey and Automath.	104
5.3	RATE definition for Monkey.	104
6.1	Data collection time, per day.	116
6.2	Number of words completed versus typing speed, per student.	117
6.3	Activity charts for sample students.	119
6.4	Summary activity chart.	120
6.5	Number of games played per minute.	123
6.6	Interactions between types of participants.	124
6.7	Who plays whom, grouped by age and gender.	125
6.8	Who plays whom, ordered by typing speed.	126
6.9	Tracking learning in sample students.	127
6.10	Change in typing speed.	128

6.11	Exit poll.	129
6.12	“How much did you enjoy the match?”	130
6.13	“How hard was the match?”	130
6.14	Post-study survey.	131
7.1	Basic control architecture.	139
7.2	Playground behavior.	142
7.3	Command probability.	143
7.4	Game behavior.	144
7.5	Neural network architecture.	148
7.6	Average typing speeds of players.	149
7.7	Improvement during training.	151
7.8	Correlation between trainers and best trainees.	152
7.9	Correlation between populations of trainers and best trainees.	153
8.1	Distance between words in feature space.	157
8.2	Selection and reproduction.	160
8.3	Exploitation and exploration in feature space.	161
8.4	Sampling illustration.	167
8.5	Sample domain coverage chart.	170
8.6	Domain coverage charts for sample users.	171
8.7	Word length vs typing speed.	172
8.8	Feature correlation with typing speed.	173
8.9	Correlation coefficients.	176

9.1	Sample IDsigns.	186
9.2	Sample playgroup graph.	191

Chapter 1

Introduction

In the last two decades, the rise in popularity of personal computers has spawned a new and burgeoning market for educational software. With the introduction of computers into schools, this software has infiltrated classrooms. Schools are being “wired” at a rapid rate, giving teachers and students direct access to the Internet. Yet despite all the vast and varied hardware and software, the order of magnitude improvement in student performance that many expected as a result of computer-based interactive learning systems is not evident.

There are many reasons for this shortcoming. Some believe that the educational software market has exploded too fast, without enough pedagogy behind the software or developmental psychology supporting schools’ technology integration decisions [Healy, 1999]. “Once net connections are established,...many teachers find a shortage of quality software tools and curricula to make use of them.” [Bruckman & DeBonte, 1997] And there are practical issues as well.

“Computer software and hardware become obsolete every 30 months, too swift a change for most schools to handle economically.” [Gonzalez, 2000]

Most learning systems have not been successfully deployed in practical environments, in spite of expensive resources and years of research. Kinshuk and Patel [1997] cite two primary reasons for this failure: (1) the underlying methodologies for developing most learning systems were not designed from an educational viewpoint, and (2) the development of most learning systems has left out the needs of teachers and students.

Indeed, John Anderson’s original motivation was “to learn more about skill acquisition rather than to produce practical classroom results.” [Anderson et al., 1995] In a paper outlining lessons learned from working on intelligent tutoring systems (ITS) for over 10 years, Anderson et al. [1995] list several reasons why their tutors were not put into general use in classrooms, including: “there was never any attempt on our part to address the curriculum that educators wanted to teach.” As well, “the systems that we developed were inflexible in the way they had to be used and gave teachers no ability to tune the application of the tutors to their own needs and beliefs about instruction.”

However, “there is no mention of any existing ITS in the literature which allows the teaching community to contribute towards the development of an ITS without starting the design process from scratch.” [Kinshuk & Patel, 1997] Primarily, this is because “knowledge-based educational software, such as intelligent tutoring systems, have historically been large, self-contained programs with specialized platform requirements.” [Suthers & Jones, 1997]

Clearly, the field of interactive learning systems (ILS), which includes instructive intelligent tutoring systems as well as constructive environments, has problems. The work presented in this thesis responds to three specific issues:

1. *The practical needs of schools.* It is impractical to ask schools to install fancy educational systems and keep up with hardware and software upgrades. Additionally, the hardware at many schools is several generations behind the equipment used in research labs.
2. *The curricular needs of teachers.* Educational practitioners should have an active role in building educational software, both for experimental and commercial implementations.
3. *The overlapping system needs of researchers.* Many researchers with a variety of backgrounds and goals are studying human learning and technology, and each group is building their own complete system, despite the fact that many underlying system components are the same.

1.1 Contribution

The main contribution of this thesis is to put forth a practicable interactive learning system designed to support the types of activities, experiments and data collection which are common to the ILS field, while answering needs that have not been addressed satisfactorily in the past. The thesis presents a prototype system called CEL (*Community of Evolving Learners*) [Sklar & Pollack, 1998;

Sklar & Pollack, 1999; Sklar & Pollack, 2000b] which provides an environment that is:

- *accessible*, because it can be reached by schools with minimal hardware capabilities and no specialized software installation requirements;
- *flexible*, because it can host a variety of curricular and experimental activities; and
- *extensible*, because it offers a shareable framework to which others can add their own activities.

The basis of this work is in computer science, not education, psychology or cognitive science; so the purpose here is not to set forth a new pedagogical example. On the contrary, the goal is to establish a platform that others with research interests in human learning can use to define and implement their own studies. The CEL system is specifically engineered for re-use, so that it can be shared by others to host a variety of activities, without requiring others to build an entire interactive learning system “from scratch.”

CEL is implemented on the Internet because in order “to reduce cost...and enable greater collaboration...educational materials should be shareable between diverse applications across the Internet.” [Suthers & Jones, 1997] As well, Internet learning communities offer several advantages over traditional educational software [Kinshuk & Patel, 1997; Stanchev, 1993]: many-to-many communication, place independence, time independence, multi-media support and computer-mediated interaction. Inside CEL, users engage in multi-player educational games

because “play appears to be a universally accepted mode of learning.” [Amory et al., 1998] Also, games are “attractive to many children, and exploratory and interactive in nature.” [Klawe & Phillips, 1995] Multi-player activities are implemented because much research has shown group learning to be highly effective [Johnson & Johnson, 1989; Slavin, 1992; Slavin, 1995].

CEL is accessible, taking advantage of the Internet in two ways. First, the technology behind the Internet means that the system can be used by people from all over the world with varying hardware and software capabilities. Second, the distributed nature of the Internet means that learners can participate anonymously, allowing students to succeed and fail *incognito*, without the normal social pressures of a traditional classroom setting.

CEL is flexible, having the ability to host many types of games and collect many types of data, supporting teachers and researchers by enabling a variety of curricular activities and experiments. The games in CEL may be synchronous, where players take turns, or asynchronous, where play happens in pseudo real-time. Games in CEL may be collaborative or competitive. Players may participate openly or secretly, with each other and/or with software agents. Some of the curricular activities that have been hosted include: a spelling bee, an anagrams game, typing races, a collaborative building activity and arithmetic exercises. Some of the types of experiments that can be conducted include:

- comparison of methods for defining curricular paths within a knowledge domain
- analysis of competitive versus collaborative settings for an educational game
- study of interactions in human-human versus human-agent encounters as opponents in competitive, or partners in collaborative, activities
- comparison of different user interfaces for the same underlying engine

CEL is extensible, allowing others to implement and host their own activities and collect their own data, while tapping into our client-server architecture, facilitating communication through our system server, handling data with our database manager and gaining access to a common infrastructure and user base. This design enables rapid creation and dissemination of additional games and abstracts away complex system building issues such as client-server communication, player distribution and synchronization, and data capture and storage. This means, for example, that contributors could be teachers or non-technical researchers, who could lay out new pedagogical activities and then work with undergraduate programmers to extend our model and implement the new activities.

Our target user group is primary school children, therefore we pay particular attention to issues of privacy and safety. CEL offers an alternative to traditional learning environments, breaking physical barriers of classroom walls and linking students with similar abilities but diverse ages, genders and locations.

1.2 Outline

This thesis presents the CEL environment, detailing the system architecture, explaining the types of activities that can be hosted by CEL and the types of data that can be collected in CEL, and demonstrating the accessibility, flexibility and extensibility of the system. The thesis is organized into nine chapters.

Chapter two provides background in interactive learning environments, identifying elements of these systems that are pertinent to researchers and describing the types of data that is generally collected and analyses that are typically performed. The focus is on educational games and Internet learning communities.

Chapter three introduces the CEL system, from a user's point of view. An overview of the environment is given, site components are explained and terminology is defined that will be used throughout the thesis.

Chapter four gives a detailed description of the modular system architecture, focusing on its accessibility and extensibility. The basis is a client-server model. Particular attention is paid to the needs of the client since the user base for CEL includes school children with low-end computers — slow network bandwidth and limited memory.

Chapter five discusses the data that is gathered inside CEL and describes the databases that are in use. The emphasis is on the flexibility of the data collection module.

Chapter six reports on pilot testing in which CEL was used in a public primary school by fourth and fifth grade children. During the early phase of this

period, the architecture of the system was adjusted in response to various issues raised at the client site. The remainder of the period was spent collecting sample data and this chapter closes by demonstrating the types of analyses that could be performed on CEL data, using the pilot study as an example.

Chapter seven details the use of software agents as artificial learning partners who inhabit CEL in order to sustain the community by maintaining a “live” presence at all times. CEL allows flexibility in the choice of a control mechanism for these agents. The agents described here are controlled by neural networks that were trained using a new approach, based on human usage data gathered during pilot testing.

Chapter eight outlines a method used for domain knowledge engineering, exhibiting CEL’s flexibility in the choice of a domain delivery mechanism. The method presented is a novel approach in which curricular paths are allowed to emerge as students interact with CEL with the goal of achieving individualized domain coverage that adapts on-line to the needs of each student as each student learns. Note that this work is likely the first to take an evolutionary approach to problem selection in an interactive learning system.

The thesis concludes with a specific defense of the three initial claims — that CEL is accessible, flexible and extensible. Finally, future endeavors are discussed, wherein the current capabilities of the system will be expanded, and new directions will be explored.

Chapter 2

Background

No matter what we do, a huge infusion of technology is coming to education. It doesn't matter if it works or not, whether we make mistakes or not. It's coming because so much money is behind it. And because that infusion of technology is inevitable, it would be nice to start adding some new perspectives about technology in the schools. It's just possible our decisions about technology in schools are not being guided by the instincts of our best teachers. Right now, we run the risk of being blinded by science. [Snyder, 1994]

In general, there is a disconnect between educational researchers and practitioners [Reeves, 1999], and this divide is increased when the topic of research is educational software, or interactive learning systems (ILS). Often the builders of these systems are either computer scientists, not trained in education, or educators, psychologists and/or cognitive scientists, not trained in system building.

There is clearly a need to connect the work of education practitioners with that of human learning researchers and builders of learning systems. This chapter examines literature on interactive learning environments, in an effort to unify the interests of the different groups since the motivation behind the CEL system is to

provide a platform that could be used by any of these groups for experimentation or in support of curricular activities. Because the CEL environment involves Internet-based educational games, the emphasis in this chapter is on educational games and Internet learning communities.

This chapter is organized as follows. Initially, two questions are addressed: What are interactive learning systems? How are interactive learning systems evaluated? Then several interactive learning studies, at varying levels of maturity, are discussed. Particular attention is paid to the following four elements:

1. the types of issues addressed in the ILS field,
2. the kinds of environments supported and system components built,
3. the forms of data collected, and
4. the various testing and analyses performed.

Each of these elements has contributed to the composition of the CEL platform. The chapter closes with a summary that highlights the specific ways in which CEL is designed to address these four elemental requirements.

2.1 What are interactive learning systems?

Ultimately, all learning is interactive in the sense that learners interact with content to process, tasks to accomplish, and/or problems to solve. However...I refer to a specific meaning of interactive learning as involving some sort of technological mediation between a teacher/designer and a learner. [Reeves, 1999]

There are two major approaches taken in the field of interactive learning systems (ILS) [Reeves, 1999]:

1. *instructive* and
2. *constructive*.

The difference can be described as follows: students learn “from” instructive systems; students learn “with” constructive systems.

2.1.1 Instructive learning systems

Instructive learning systems have a basis in educational communication theory, where researchers look to find the best ways of communicating new ideas to learners. Traditional computer-aided instruction (CAI) applications and intelligent tutoring systems (ITS) are examples of this type of system.

ITS's grew out of fixed-path, drill-and-practice CAI applications. Seminal work in the ITS area began with frame-based tutoring systems [Brown & Burton, 1978], exhibiting such desirable characteristics as providing traces of problem-solving sessions, customizing for individual users, dynamically selecting what to do next and coaching users at opportune times [Clancey, 1986].

Early work combined some or all of these features, each emphasizing issues such as memory modeling [Schank, 1981; Kolodner, 1983], construction of rules [Anderson, 1982], and representation of students' misconceptions [VanLehn, 1983; Soloway et al., 1981]. Many of these ideas were developed into systems tested in laboratories, classrooms and work places [Koedinger & Anderson, 1993; Schank & Cleary, 1995].

Subsequent work has continued to explore these areas in more depth. Student modeling has been aided by statistical techniques [McCalla & Greer, 1994; Conati & VanLehn, 1996; Beck, 1997; VanLehn et al., 1998] as well as artificial intelligence methods like case-based reasoning [Shiri-A. et al., 1998].

2.1.2 Constructive learning systems

Constructive learning systems are based in cognitive psychology, where the computer is seen as a cognitive tool or learning partner. *Constructivism* originated with Jean Piaget and states that knowledge being acquired is built by the student, rather than being supplied by the teacher [Gruber & Voneche, 1977].

Seymour Papert suggested *constructionism* as an expansion to constructivism, postulating that students learn better when they are actively engaged in building something external to themselves. Their construction could be a physical object, like a castle, or a virtual object, such as a room in a virtual world [Papert, 1991]. In constructionist environments, students are able to explore ideas for themselves without having to stick to a fixed curriculum [Papert, 1993]; and students at all levels of ability are provided with opportunities to learn.

For example, Papert developed LOGO [Papert, 1980], a simple computer language that children can use to program graphics environments. In LOGO, the cursor is a “turtle” that moves around the screen based on commands given by the student. Later, Mitch Resnick and Uri Wilensky developed Star*LOGO [Resnick, 1997], which builds on the LOGO setting and adds a dimension of parallelism. Instead of having one turtle, the programmer is given many, even thousands, of turtles. Students can experiment with adaptive behavior, by giving a group of turtles the same commands and observing what happens as the members of the group perform the commands and in doing so interfere with each other.

2.2 How are interactive learning systems evaluated?

There is no simple standard for evaluating the effectiveness of interactive learning systems, either instructive or constructive. Mark and Greer [1993] review methodologies commonly used, describing two general categories of assessment¹:

1. *formative* — used to assess the design and behavior of a system in-progress, generally performed by computer scientists
2. *summative* — used to assess the effectiveness of a completed system, generally performed by educators and/or psychologists

¹following from Littman and Soloway [1988]

Table 2.1 lists common components of interactive learning systems and some corresponding evaluation criteria.

Table 2.1: Components of and evaluation criteria for ILS.

component	evaluation criteria
<i>domain knowledge</i>	<ul style="list-style-type: none"> • accuracy
<i>teaching component</i>	<ul style="list-style-type: none"> • range of instructional method(s) offered • level of adaptability • degree to which instruction is based on educational and psychological testing
<i>user interface</i>	<ul style="list-style-type: none"> • comparison of multiple user interfaces for the same underlying engine
<i>student knowledge</i>	(note that the same criteria are used to evaluate standard educational and/or psychological tests): <ul style="list-style-type: none"> • <i>validity</i> — does the test show evidence that it measures what it says it measures? • <i>reliability</i> — are multiple results for the same subject consistent? • <i>objectivity</i> — is the test administered and scored the same way for every participant? • <i>standardization</i> — can results be translated into a meaningful representation of student performance?
<i>system adaptivity</i>	<ul style="list-style-type: none"> • comparison of interactions at different skill levels
<i>control component</i>	<ul style="list-style-type: none"> • system performance measures (e.g., speed)

(Source: interpretation of [Mark & Greer, 1993])

The techniques for performing assessments vary depending on which component is being evaluated, where in the system development cycle the evaluation is being performed and who is performing the evaluation. *Pilot testing* often occurs late in formative evaluation, bridging the gap to summative evaluation. There are three methods of pilot testing: *one-to-one*, which is performed early in the development cycle, with one student, teacher or researcher providing feedback; *small-group*, which is performed later in the development cycle, with a small

group of students (or teachers) providing feedback; and *field*, which is performed near the end of development, emulating experimental conditions with teachers and students in a “live” (school) setting.

The list of criteria in table 2.1 is of primary concern during formative evaluation. Other techniques are more pertinent during summative evaluation. In *criterion-based evaluation*, a general list of guidelines is developed and systems are evaluated based on their adherence to these guidelines, for example, program construction, behavior and characteristics. While developing specifically relevant criteria is not an easy task, this method may prove useful in formative assessment and in comparing different systems. With *expert knowledge and behavior* assessment, system performance is compared to that of a human expert performing the same task. One famous example is the Turing test [Turing, 1963]. Software systems may be subjected to a standard *certification* process, just as human teachers are, perhaps through careful examination by qualified human experts. In *sensitivity analysis*, the responsiveness of a system is tested on a variety of different user behaviors. This may be particularly useful for evaluating system adaptivity. After system development and pilot testing are complete, *experimental* research begins. The conditions should be the same as those during the field testing phase. Figure 2.1 illustrates the progression from system development through experimental research.

In reviewing the pilot testing and experimental literature, two types of evaluations are common: (1) comparison of pre- and post-tests, to measure changes in student performance, and (2) analysis of on- and off-line surveys, to deter-

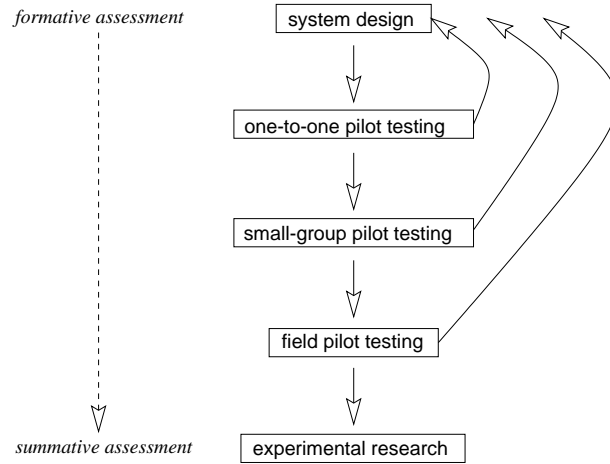


Figure 2.1: System development cycle

mine students' interest in a system. This reflects the fact that most systems are assessed based on learning and/or motivation.

2.3 Interactive learning studies

As mentioned earlier, there are many diverse groups studying interactive learning, each with its own interests, goals and methods of analysis. This section samples some of the interactive learning literature, describing several projects at varying levels of maturity (see figure 2.1).

First, the experimental work of two prominent researchers is outlined: Thomas Malone, a psychologist who examined motivation in educational games, and John Anderson, a cognitive scientist who looked at skill acquisition in human learning.

Second, three field studies performed using Internet-based systems are dis-

cussed. The KIE system [Bell et al., 1995] is designed to help students learn to integrate knowledge when facing scientific problems. The other two systems mentioned are educational MUD's². MUD's were introduced in the late 1960's, originating with off-line text-based role-playing adventure games. Probably the most famous was called *Dungeons and Dragons*. The first digital versions came on-line in the late 1970's, on Arpanet [Bruckman, 1997], and since then MUD's have been growing in popularity. Some have used the MUD paradigm to build educational applications [Fanderclai, 1995; Gordon & Hall, 1998]. The educational MUD's discussed here are Pueblo [Walters & Hughes, 1994] and MOOSE Crossing [Bruckman, 1997].

Third, five small-group studies are detailed. The first, describes an Internet-based system called MANIC [Stern et al., 1997] that is designed to deliver course materials effectively. The remainder discuss educational games: Counting on Frank [Klawe & Phillips, 1995], Phoenix Quest [Klawe et al., 1996], ScienceSpace [Dede et al., 1996] and Zadarh [Amory et al., 1998].

Finally, two system design reports are summarized: Belvedere [Suthers & Jones, 1997] and CoVis [Pea, 1993]. Both are Internet-based systems used to support collaboration in science learning at the high school level.

2.3.1 Experimental research: motivation

Many believe that the secret to education is motivating the student. Researchers in human learning have been trying to identify the elements of electronic environ-

²Multi-User Dungeon/Dimension/Domain

ments that work to captivate young learners. In 1991, Eliot Soloway wrote “Oh, if kids were only as motivated in school as they are in playing Nintendo.” [Soloway, 1991] Two years later, Herb Brody wrote: “Children assimilate information and acquire skills with astonishing speed when playing video games. Although much of this gain is of dubious value, the phenomenon suggests a potent medium for learning more practical things.” [Brody, 1993]

Thomas Malone is probably the most frequently referenced author on the topic of motivation in educational games. In the late 1970’s and early 1980’s, he conducted comprehensive experimental research to identify elements of educational games that made them intrinsically motivating. He highlights three characteristics: challenge, fantasy and curiosity.

Challenge involves games having an obvious goal and an uncertain outcome. Malone recommends that goals be “personally meaningful”, reaching beyond simple demonstration of a certain skill (such as being able to solve a multiplication problem or spell a word). Instead, goals should be intrinsically practical or creative, like solving a multiplication problem in order to compute the cost of lemons needed to make a gallon of lemonade for a lemonade stand, or spelling the words in the marketing copy that will advertise the lemonade stand.

Malone emphasizes that achieving the goal should not be guaranteed and suggests several methods for providing this uncertainty:

1. variable difficulty level — this can be determined automatically by the system or manually by the player;

2. multiple goal levels — this can include performing a task correctly and also quickly, so that once the player has learned how to do the task right, s/he will then be motivated to learn to accomplish the goal with greater speed;
3. hidden information — as the learner progresses at the task, more information can be revealed, for example by making the task simpler when players take longer to find solutions;
4. randomness — an element of surprise or risk may engage a learner, e.g., games that involve gambling.

Fantasy is a feature that is designed to enhance the fun of learning. Fantasy can be intrinsic or extrinsic. Extrinsic fantasy involves overlaying some kind story on top of the learning task, for example allowing the player to move around a baseball diamond by providing correct answers to arithmetic problems. Intrinsic fantasy implies that the skill being learned is inherent in the problems presented, for example teaching a player about Cartesian coordinates by letting him move around in a grid space.

Curiosity provides “novel and surprising” elements, but these should not be incomprehensible to the learner. There are two forms of curiosity elements — sensory and cognitive. Sensory refers to audio and visual effects. Cognitive refers to surprises within the game content. These should also be constructive, moving the learner toward the goal, not distracting him from the goal.

Malone makes an important distinction between *toys* and *tools*. He defines toys to be systems that exist for their own sake, with no external goals; in

contrast, tools are systems that exist because of their external goals. Good games are difficult to play, in order to increase the challenge provided to the player. Good tools should be easy to use, in order to expedite the user's external goal. We can infer that good educational games should encompass elements of both. There is an external goal — for the user to learn how to perform a given task — and the learning process should be made as easy as possible. At the same time, the learner should be challenged during the learning process.

The simplest educational games take old-fashioned tutoring systems and surround them with extrinsic motivational features like attractive multi-media special effects. However, according to Mitch Resnick, building an animated world around a series of puzzles, all having correct answers, imposes an artificial environment and reduces the educational encounter to a fact-learning experience. “Pedagogically, children will learn more if the acquisition of knowledge is made integral to the game.” [Brody, 1993] The E-GEMS group at the University of British Columbia agrees, promoting “the playing of games as an integral component of [mathematics] learning.” [Klawe & Phillips, 1995]

With a rise in computer-supported collaborative learning environments, many educational games are moving into multi-player modes. McGrenere [1996] reviewed much of the literature and summarized her findings with a series of guidelines for designing educational multi-player games: provide for challenge, fantasy, curiosity and creativity; design the learning task carefully; allow learner control; allow for communication possibly through multiple modalities (audio, video, text messages); provide instant update of the game space; and provide for

awareness through the use of various views and color. These recommendations clearly follow on Malone’s research, while promoting constructivist environments as well.

2.3.2 Experimental research: skill acquisition

John Anderson is one of the most renowned early researchers in instructive interactive learning systems. His overarching research question can be summed up as follows:

One of the current controversies in cognitive science and education is whether it is possible to take a complex competence, break it down into its components, and understand the learning and performance of that competence in terms of the learning and performance of the components. [Anderson et al., 1995]

Anderson’s work is based on the ACT theory, which is a theory of skill acquisition that began with ACT* [Anderson, 1982] and became ACT-R [Anderson, 1993]. The basic principles of ACT include distinguishing between declarative knowledge and procedural knowledge (i.e., stating a fact and knowing how to use that fact), relating this knowledge to task goals and converting the declarative knowledge into production rules to help achieve the task, and strengthening the knowledge through practice. In general, Anderson defines computer-based tutoring systems using a “model-tracing” approach wherein a correct behavior is modeled as a set of production rules and it is intended that the student should follow this model in his learning.

Anderson and his colleagues developed three major systems: geometry tutor, algebra tutor and LISP tutor. Many experiments were performed over a number

of years with these systems. Their summative analysis included four components:

1. production practice — this is a measure of how often students apply the relevant production rules,
2. within-problem practice effects — this is a measure, over time, of how a student improves his application of the appropriate production rules,
3. acquisition factor — this is a measure of how well students performed with new rules introduced during a lesson, and
4. retention factor — this is a measure of how well students retained rules learned during earlier lessons.

They performed formative experiments to determine the appropriate form and level of feedback that should be given to students and the amount of error correction that should be required. They compared several levels, including no feedback, feedback provided immediately by the system and feedback provided on demand (only when requested by the student). They gave students a fixed set of exercises to complete with one version of the system. They administered both on-line and paper-and-pencil post-tests to determine which feedback mechanism was most effective. The version that provided immediate feedback and immediate error correction gave the best results.

They also carried out some tests to determine what the content of feedback messages should be. Measuring the speed with which students provided correct answers showed that feedback messages containing some explanation of

a student's error were significantly more effective than feedback which simply indicated that the student had made an error. However, post-testing, after the session was over, did not result in statistically significant differences, indicating that providing more explanation during initial learning did not appear to offer longterm learning benefits.

One issue they point out is that creators of tutoring systems need to pay attention to user interface design early on, rather than building the underlying tutoring mechanism first. It is important to identify the skills being learned and the environment in which these skills will be used, once acquired. The user interface should be designed so that transferral of these skills from the learning environment to the "real world" will be smooth.

Practical deployment of these systems offered further learning opportunities for the researchers. They stated [Anderson et al., 1995] that there was no attempt on the part of the researchers to address the curricular needs of teachers. There was no larger educational objective — the post-tests were the only measure of success. There was little understanding of how to deploy the software tutors in the classroom, and the tutors were not flexible, so teachers could not customize the tutors to the meet the needs of their classrooms. As a result, the researchers began to develop working relationships with public schools and classroom teachers.

2.3.3 Field study: KIE

*KIE*³ (*Knowledge Integration Environment*) is designed to help students with science learning and is focused on bringing students and evidence together to solve problems. This system includes tools like an electronic space for taking notes, an on-line discussion facility and a knowledge integration coach that provides hints to students as they work together to answer scientific questions. A more recent version of this system is called the *Web-based Integrated Science Environment (WISE)*⁴.

KIE stresses the integration of science knowledge, arguing that today, science education is often too abstract. Students are exposed to a broad scope information and they do not gain in-depth understanding of this knowledge. The pedagogical basis of the KIE project lies in a “scaffolded knowledge integration framework”, which aims to teach students to reconcile scientific models with intuitive observation and distinguish between technical and colloquial use of scientific terminology. KIE defines activities and software tools designed to help students learn to use the Internet effectively for research, critique evidence and integrate new knowledge. The overall goal is for students to gain an integrated understanding of science topics.

A pilot study was conducted with KIE in which students were given the following question to answer: “How far does light go?” 165 eighth grade students participated in a school setting. They worked in pairs, sharing 16 computers. The

³<http://www.clp.berkeley.edu/KIE.html>

⁴<http://wise.berkeley.edu/WISE/welcome.php>

study began with students providing their intuitive answer to the question, prior to doing any research. Next, students reviewed evidence on the web to support or refute their intuitions. Third, the students collaborated to come up with their own empirical evidence, and then they posted that evidence on the web for others to share. Fourth, they built a scientific argument, by integrating the evidence found in the second and third steps. Fifth, they presented their arguments to the class and discussion ensued. Finally, they took a post-test and provided informed answers to the original question. Overall, the study demonstrated that the KIE approach was feasible and the study suggested and motivated improvements to the system, both in terms of software and curriculum.

The technical components of the system include: a web browser, an HTML editor, email software, a system navigation tool for selecting components, an on-line notebook, a networked evidence database (containing information from steps two and three, above), a multi-media discussion tool, a teacher tool for designing activities, and an on-line coach.

The overall goal of the KIE system is to help make the Internet a partner in education and to teach students lifelong skills for science exploration.

2.3.4 Field study: Pueblo

Pueblo⁵ was originally called *MariMUSE* and was used at a summer camp for primary school children in Arizona in 1993. Two years later, the MUD was given its current name. The focus in Pueblo is on learning through writing,

⁵http://pcacad.pc.maricopa.edu/Pueblo/index_frame.html

programming and simulation. Participants create their own virtual world and take on new identities, using the language of the MUD to invent and describe places and creatures, defining appearances and enacting behaviors. Researchers found that Pueblo served to draw otherwise uninterested children into literacy activities and that the paradigm helped to break down traditional classroom and social boundaries.

The motivation behind Pueblo is to be able to link experts and learners via the Internet. Typically, a classroom contains one expert (a teacher) and many learners (students). The Internet facilitates linking many experts with students and allows people to participate anonymously.

Researchers were interested in answering several questions with regard to the system. Can elementary students use a MUSE environment? Is there any indication that writing skills develop as a result of using Pueblo? Do primary school teachers believe in the system?

The data collected in the Pueblo system includes demographic information (gender, age and race of participants), experimenters' observations, anecdotal information, session transcripts, daily evaluations and journal entries.

Two years after the introduction of the Pueblo project, Billie Hughes highlighted some on-going challenges in making the system useful in a school setting [Hughes, 1995]. These could be considered recommendations for any type of system designed to be used in schools. Key issues are:

- Teacher start-up — School implementations entail use of local staff for solving technical installation problems, who must be trained. As well, there are training issues involved in introducing teachers to the system.
- Student start-up — Early use of the system is difficult because students have to learn many commands in order to just get around in the environment, before being able to really create anything.
- Toolbox creation — It is helpful to have a toolbox of generic objects with generic behaviors that teachers and students can invoke when they first begin using the system, before they are ready to learn/perform low-level programming tasks.
- Help — Even after initial start-up, help is often needed, especially for users without a programming background; in addition to creation of a toolbox (above), a tutoring facility would be useful.
- Reporting mechanism — In order for this environment to be useful for teachers, there needs to be a toolbox of reporting mechanisms for extracting samples of students' work, examining students' activities with the system and showing students' progress over time.
- Reality check — Young children have trouble distinguishing the real from the virtual, and researchers and teachers need to be aware of this.

2.3.5 Field study: MOOSE Crossing

MOOSE Crossing⁶ is a constructionist environment designed for late primary and middle school students. It is a text-based MUD, however MOOSE Crossing uses a new language called *MOOSE* which was the first MUD language designed explicitly for children. MOOSE Crossing is enabled on the Internet, however it is not accessible from inside a browser. Participants must download software onto their computers and connect to the MUD using this software. Participants enter MOOSE Crossing with a user name and password, but must apply for membership off-line by sending in forms signed by the child member and her parent/guardian. Adults monitor discussion on the site, and anyone found misbehaving will be denied future access. MOOSE Crossing has been extremely well-received by researchers as probably the first widely available environment of its kind, and analysis of participation has revealed positive results similar to those found in Pueblo.

MOOSE Crossing is designed to be used from home, as part of organized after-school activities and as a classroom activity. [Bruckman & DeBonte, 1997] reports on pilot testing that was performed with five classes in four schools in three different states. “Too many factors vary among these classes to warrant a formal comparison. However, a case-study analysis reveals a number of educationally significant features.” [Bruckman & DeBonte, 1997]

Data collected during pilot testing included: transcripts of discussions with

⁶<http://www.cc.gatech.edu/fac/Amy.Bruckman/moose-crossing/>

students and teachers (both on- and off-line), objects created by children, log files of interactions and observations of researchers. Permission for collecting this data was obtained from both students and their parents. Statistics include: location of study, grade level of participants, number of participants, number of adults present to assist, length and frequency of sessions, average number of commands typed per student, average number of objects owned per student, and average number of scripts written per student.

Results are presented both in terms of overall averages, grouped averages (for each classroom) and individual averages and examples. The system is assessed on four measures: access, peer experts, free-form versus structured activity and atmosphere. Access refers to the ease with which students have access to a computer where they can log on to MOOSE Crossing.

In schools where a computer is physically located in the classroom, children are often allowed to use that computer during their free “choice” time. Bruckman [1997] states that in these classrooms, children “regularly” use MOOSE Crossing during their free time. There is no statistical data to support this claim, nor any indication as to the percentage of children who chose to use MOOSE Crossing “regularly”, nor what other computer programs are available to children at that time, etc.

Other factors were observed during the pilot testing. The presence of peer experts improved use of the system and these students served as a valuable resource for others. Some of the students’ experiences with the system were motivated by collaborative projects designated by the teacher; the structure appeared to

help keep students involved in using the system. There seems to be a mixture of attitudes and cultures toward allowing students to get up and walk around and talk to each other during computer lab time.

More comprehensive analysis and statistical results can be found in [Bruckman, 1997].

2.3.6 Pilot study: MANIC

MANIC is a system designed to intelligently and adaptively deliver course material over the WWW, using existing slides and video. The authors want to personalize the delivery according to the educational needs and learning style of individual students. There were three main goals of the project: to guide students through the course material, to provide interactive/adaptive quizzes, and to pre-fetch course material.

The system architecture is based on a client-server model. There is a server-side database that contains the slides and video. The client-side contains a web browser and plug-ins (to support presentation of audio and visual materials).

The domain being delivered is organized into a set of topics. Unlike traditional ITS's (like Anderson's LISP tutor) that pre-determine what a student will see, MANIC does not impose a strict presentation order. As well, MANIC includes multiple versions of the same slide (e.g., easy and hard).

A student model is maintained, containing scores for five measures: amount of a topic viewed, version viewed (e.g., easy or hard), access patterns, hyper-links followed to review topic (which implies some level of incomprehension or perhaps

inattention), and quiz performance. Based on the student model, two methods are used for guiding students: adaptive navigation support (as in ELM-ART [Brusilovsky et al., 1996]), which adapts which links are shown, and adaptive presentation, which adapts the content shown.

A study was performed with 15 university students. Aside from the information gathered by the student model, additional data was collected including a post-study questionnaire, primarily to ascertain if the students liked the system. 9 of the 15 students completed the survey.

2.3.7 Pilot study: Counting on Frank

[Klawe & Phillips, 1995] describes a study involving an educational game called *Counting on Frank*, which was designed for teaching math. In this study, the game was used by primary school students in a collaborative mode where small groups of learners worked together at one computer.

Games are an attractive medium for teaching math because they are appealing to children, they are exploratory and interactive in nature, and they facilitate visualization. The philosophy embraced by the researchers here asserts that the “playing of games is an integral component of mathematics learning, rather than as a way to trick students into paying attention before the ‘real teaching’ starts.” [Klawe & Phillips, 1995]

The Counting on Frank study puts forth the notion of “student as researcher”, because the students kept logs of their experiences with the system, which included bugs they found and criticisms they had. The study emphasizes

group discussion, both before and after the sessions with the system. The results are anecdotal, based on data collected in students' logs and observations of experimenters.

The authors conclude by outlining several issues highlighted by their work. First, it is beneficial to have two students work together at one computer. However, the authors offer no comparison — is their scenario better than students working alone at one computer or better than two students working together at two computers? Second, requiring students to make use of external tools while using a computer may help with knowledge transfer. Third, careful consideration must go into the design of user interfaces for education, particularly in relation to choosing a highly intuitive versus more deliberate design because students may lose opportunities for learning when using an interface that does too much. In summary: “making computer use more efficient for learners can sometimes result in less effective learning.” [Klawe & Phillips, 1995]

2.3.8 Pilot study: Phoenix Quest

Phoenix Quest is a computer game designed to encourage (especially) girls ages 10-14 to explore concepts in mathematics and language arts. [Klawe et al., 1996] describes a study which showed that girls value story line, characters, worthwhile goals, social interactions, creative activities and challenge. In comparison, boys value fast action, adventure, challenge and violence. The results are largely anecdotal.

The study took place over a 6 week period, in 40 minute sessions. The first

half of the study was spent fixing implementation problems. Approximately 120 groups of primary school children (grades 3-7) participated. There were between one and three children in a group. Overall, there were nearly the same numbers of girls as boys.

The following data was collected: log files, which included correspondence and game events, participants' ratings, researchers' observations, and interviews with teachers and some of the students. The ratings mentioned came from a post-session survey that was conducted on-line, where students rated their experience after each session according to three criterion: fun, importance and challenge.

Analysis examined session completion rate (normal or error) for both boys and girls and the average rating (from the post-session survey) per category for both genders. Gender differences were noted. No data or results on learning was presented.

2.3.9 Pilot study: ScienceSpace

[Dede et al., 1996] compares and evaluates three virtual reality microworlds that comprise a system called *ScienceSpace*. Formative evaluation compared three user interface styles. Summative assessment was performed in terms of usability and learning.

Usability was evaluated according to the following objective and subjective measures: task completion, error frequency, ratings by participants (“easy” or “hard”), rankings of interaction styles, participants' comments, and researchers observations. Learning was measured by comparing results of pre- and post-tests

of participants' knowledge.

Three multi-sensing interfaces were compared: (1) visual only, (2) visual and auditory, and (3) visual, auditory and haptic. The determination was that the more multi-sensory cues that were available, the more the students were engaged in using the system, concluding that students find virtual reality attractive for learning.

Anecdotal remarks are shared, however no statistical results are reported.

2.3.10 Pilot study: Zadarh

[Amory et al., 1998] conducted a study in which they compared four single-player games in order to determine the types of games enjoyed by students, in this case undergraduate biology students, and then used the results to design their own game. The four types were: strategy, adventure, simulation, and shoot-em up. They divided participants into racially and gender balanced groups and recorded additional demographics which included age, race and amount of computer experience.

The participants played each of the four games and afterwards completed a questionnaire. They rated the games on a scale of 0 to 4, for specific characteristics within three categories:

- | | |
|----------------------------------|--|
| 1. game aspects | funness, sound/graphics, game type, story, technology |
| 2. assessment of skills required | logic, memory, visualization, math, reflexes, problem solving |
| 3. game play | too easy, addictive, boring, too long, challenging, confusing, too difficult, illogical, difficult to play, practice makes perfect |

After this study, a new learning game was built, called “Zadarh”. This game was assessed using the same questionnaire as above. In addition, the groups completed pre- and post-tests. The results showed that adventure and strategy games were preferred (over the other types tested). They report that “[the students] also learnt something while playing,” [Amory et al., 1998] although this statement is not supported with any statistical or even anecdotal evidence.

2.3.11 System design: Belvedere

*Belvedere*⁷ is designed to assist students learning critical inquiry skills for science domains. The primary element of the system is a collaborative inquiry database, which students can access through a variety of interfaces. The database helps them keep track of the problem they are addressing, their hypotheses, evidence and references. The system also includes a sophisticated coach that can help students during the critical inquiry process.

Belvedere has five components: (1) a collaborative inquiry database, where students keep a record of their inquiry process, (2) a Java-based interface into

⁷<http://lilt.ics.hawaii.edu/belvedere/>

the collaborative inquiry database, (3) HTML interfaces which serve as a backup to the Java interface, (4) a coach designed to stimulate students, and (5) HTML reference materials designed to scaffold students. Components are implemented using Java, CGI, Lisp, HTML and SQL.

There are several educational issues at hand: lack of motivation, limited knowledge of scientific domains, inability to understand theories and arguments, particularly abstract concepts, difficulty in keeping track of debate, and lack of scientific basis in arguments. The interface is specifically designed to address each of these issues.

2.3.12 System design: CoVis

*CoVis*⁸ is geared towards forming distributed electronic communities dedicated to science learning in K-12 environments, particularly through scientific visualization. As well, the system provides links to Internet resources and several interaction devices, such as real-time collaborative environments for conversing with teachers and other students. Specific curricular activities are built into the system, such as learning about the weather using graphics tools that let users view climate maps or satellite imagery.

School-based learning communities are formed by teachers and students to support long-term collaborative projects, “allowing them to learn from one another and letting the problems to be solved dictate the knowledge that must be acquired,...frequently and purposefully crossing disciplinary boundaries.” [Gordin

⁸<http://www.covis.nwu.edu/>

et al., 1996] There are various levels of interaction in these communities that are currently in use:

1. Information resources — Examples are published work and analyzed data. This includes information provided by libraries, museums and government sites, as well as curricula and activities found on educational sites, and indices offered by a variety of sources. This information is generally available as text or hypertext with images.
2. Analysis resources — Examples are raw data and analysis tools, such as weather data and visualization software.
3. Interaction with community members — There are several categories of interaction: connecting teachers to each other, connecting students to each other and connecting parents and local communities with schools. Additionally, some sites provide connections between students of one school with teachers at another school.
4. Collaboration with community members — The vision here is to connect experts from within a local community with teachers and students in schools.
5. Publication of community's work — The result of a community's work can, if published on-line, feed back as information and analysis resources, described above.

2.4 Summary

In designing CEL to be an accessible, flexible and extensible interactive learning system, it is necessary to position CEL solidly as a viable ILS so that it can sustain the basic requirements of the field. As such, CEL must be able to support the types of issues, environments, data collection, testing and analysis put forth throughout this chapter.

CEL responds to the most prominent issues in the ILS field of study as follows:

- motivation in learning — Using Malone’s definition, CEL is both a tool and a toy. It is a tool for researchers and teachers, but the games inside CEL should be considered toys by the students that use them. They should be intrinsically motivating, providing elements of challenge, fantasy and curiosity. The sample activities outlined in chapter 3 take these concerns into account.
- acquisition and synthesis of complex skills — Following from Anderson’s work as well as the three systems that support collaborative scientific inquiry (KIE, CoVis and Belvedere), it is important that CEL be able to handle simple domain elements and support individual acquisition of these elements as well as integration of multiple elements into complex groupings. The nature of CEL promotes flexible definition of domains, elements within these domains, groupings of elements and actions concerning the elements. The message language described in chapter 4 and the data handling tech-

niques discussed in chapter 5 outline the mechanisms in CEL that address this issue.

- effective use of Internet technology at varying age levels — The philosophy of CEL is to adjust to the broadest standard software environment that is sensible in order to service the most number of participants while still delivering a useful system. This means that the hardware requirements should be minimal, an issue which is discussed in chapters 4 and 6. As well, the user interface, described in chapter 3, is designed to be usable by children as young as age 8. Although this means that some of the nomenclature (e.g., “playground”) may seem silly to older children, the environment will be understandable by a larger segment of the population.
- productive integration of learning systems in schools — The startup issues highlighted by [Hughes, 1995] as well as installation and upgrade concerns mentioned by [Gonzalez, 2000] are addressed by the accessible feature of CEL. The basic CEL “playground” interface (see chapter 3) requires a minimum of instruction to use, and so startup time for both teachers and students is short. Because CEL runs inside a standard web browser, it requires no special software installation, hence no special upgrades.

The flexibility of CEL enables it to support the different types of environments that are popular in the ILS field. This includes both instructive and constructive activities. While many ILS environments are single-user, CEL activities are all multi-player games (currently two players). Note that in CEL

both players need not be human; chapter 7 discusses the use of software agents as artificial game partners or opponents. ILS environments may be competitive or cooperative; the latter is more typical, since collaborative learning is highly touted by today's experts and competition in education, especially in the United States, is highly controversial [Kohn, 1986]. Chapter 3 illustrates competitive and collaborative games in CEL.

Quite a few ILS's allow direct on-line communication between participants. In most Internet learning communities, communication is explicit, e.g., via a collaborative workspace (as in KIE) or through the English-like language of a MUD (as in Pueblo and MOOSE Crossing). In CEL, open communication is not permitted, in order to protect young participants. Instead, CEL members interact with each other and/or with software agents, using the "language" of the games they are playing:

a game like, say, chess has highly formalized signs and rules; the 'language' of chess may be exhaustively described by logical syntax, without the fluidity and uncertainty of human language. So such games form an oversimplified analogy to human conversation. [Cherry, 1978], p.252.

This restricted mode of communication not only serves to protect the privacy of young participants but also offers two additional benefits. First, simple software agents can interact in CEL and pass a minimal Turing test, because the normal complications of natural language are avoided. Second, while the absence of open conversation diverges from typical computer supported collaborative learning (CSCL) systems, CEL provides the opportunity to explore non-conversational collaborative learning, which e.g., could support learning partnerships between

members who do not speak the same language.

The common components to most interactive learning systems include a user interface, a student model, a communication facility (for multi-user environments) and a control mechanism. Chapter 4 describes each of these components, which are all implemented in CEL.

The data gathered during ILS studies is typically in the form of oral discussions, video tapes, paper-and-pencil surveys and tests, on-line surveys and tests, on-line system products and on-line session logs. This data comes from (objective) pre- and post-tests, (subjective) on- and off-line surveys (where typically users are asked to rate various features on a numeric scale), user demographic questionnaires (including information like age, gender, race, location and computer experience), and observations of experimenters, teachers and students. CEL supports on-line collection of user demographics (through a login facility), surveys, system products and session logs. On-line testing is not currently implemented, but would be quite feasible in future work. Chapter 5 describes the on-line data capture and storage methods used in CEL. Of course, experimenters are free to employ any off-line methods they choose. For example, during the pilot testing outlined in chapter 6, we took video footage and administered an off-line survey.

The types of analyses performed on this data is obviously geared toward the particular goals of individual studies. However, there is often some overlap and we identify five common categories of analysis and the types of questions asked in each category:

1. *activity* — what are participants doing with the time they spend using a system? what types of activities are they involved in?
2. *coverage* — how much of the knowledge domain have students covered in their interactions with the system?
3. *learning* — how much have the students learned in their interactions with the system?
4. *interaction* — with multi-user systems, or systems involving use of software agents, how much have students interacted with other users or software agents? have these interactions changed students' activity levels? have these interactions affected learning? who have the students interacted with?
5. *interest* — do the participants enjoy using the system? are they motivated to participate? do they find elements of fantasy, curiosity and/or challenge in the system?

Table 2.2 shows the relationship between the types of data collected and the types of analyses performed.

Table 2.2: Relationship between data and analysis.

	activity	coverage	learning	interaction	interest
pre-tests			×		
post-tests			×		
on-line surveys					×
off-line surveys					×
session logs	×	×	×	×	×
observations	×			×	×
system products	×	×	×	×	×

In the field of ILS, it appears that the most common testing reported in the literature is field testing. There are very few comprehensive experimental results published, particularly with Internet environments, as the technology is relatively recent. The analyses performed are sometimes statistical and more often anecdotal. Chapter 6 describes field testing done with the CEL system and shows examples of the types of analyses that could be performed on the data collected in CEL.

Most ILS systems need tools with which to examine the data that is collected on-line. These should be usable by researchers as well as teachers. Future work with CEL involves creating a set of analysis tools, particularly for teachers.

Chapter 3

An Overview of CEL

This chapter provides a tour of the CEL environment from a user's point of view. Terminology specific to CEL is introduced. We highlight the features of the system that make it accessible for participants and flexible and extensible for contributors.

3.1 A brief tour of CEL

CEL is located on a free web site and is open to anyone with Internet access and a Java-enabled browser. Netscape is currently the only browser fully tested. We have also performed basic testing with Internet Explorer. CEL has been tested on Windows-95, Macintosh and Linux platforms. Figure 3.1 shows the home page for CEL, located at: <http://www.demo.cs.brandeis.edu/cel>.

Students log into CEL with an individual user name and password (see figure 3.2.a). In order to maintain the levels of anonymity and privacy that CEL

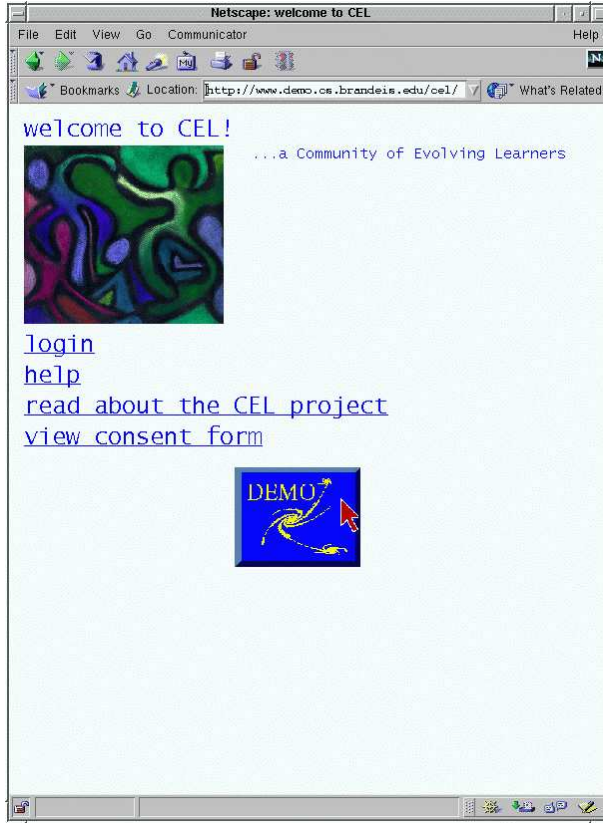
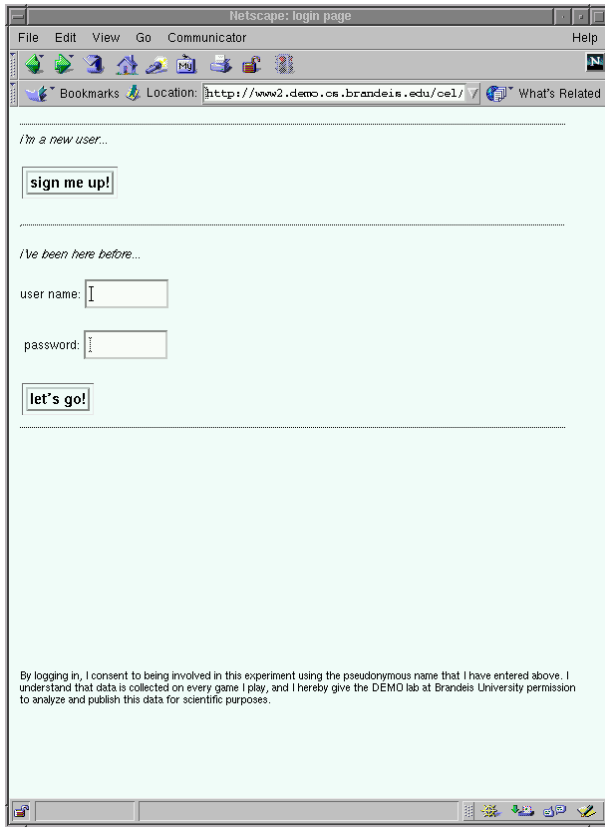


Figure 3.1: CEL Home Page.

demands, the user name (and password) are never shown to others. Unlike other virtual communities where users interact directly in open environments, CEL users only communicate indirectly — the content of each game serves to link players and participants are identified in the system solely by graphical icons. These icons are called *IDsigns* and users create their IDsigns themselves (see figure 3.2.b and section 3.2).



(a) Login screen.

(b) Sample IDsign.

Figure 3.2: Logging in to CEL.

After logging in, students are shown a simple menu page, containing a list of available activities. Clicking on a game icon selects that activity (figure 3.3).

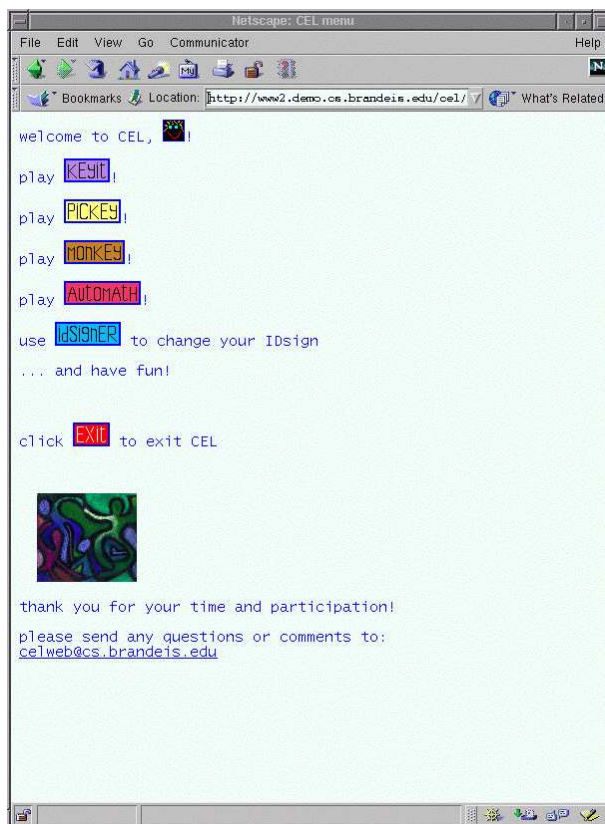


Figure 3.3: The CEL menu.

Next, users are placed in an open *playground*, a page that contains a matrix filled with IDsigns belonging to other users who are currently logged into CEL and are playing the same game (figure 3.4). These are a user's *playmates*; together they comprise a user's *playgroup*.



Figure 3.4: The CEL Playground.

By clicking on a playmate's IDsign, a student invites a playmate to join her in a *match* (see figure 3.5). Depending on the type of game being played, the match could be collaborative or competitive, free-play (asynchronous) or turn-taking (synchronized).

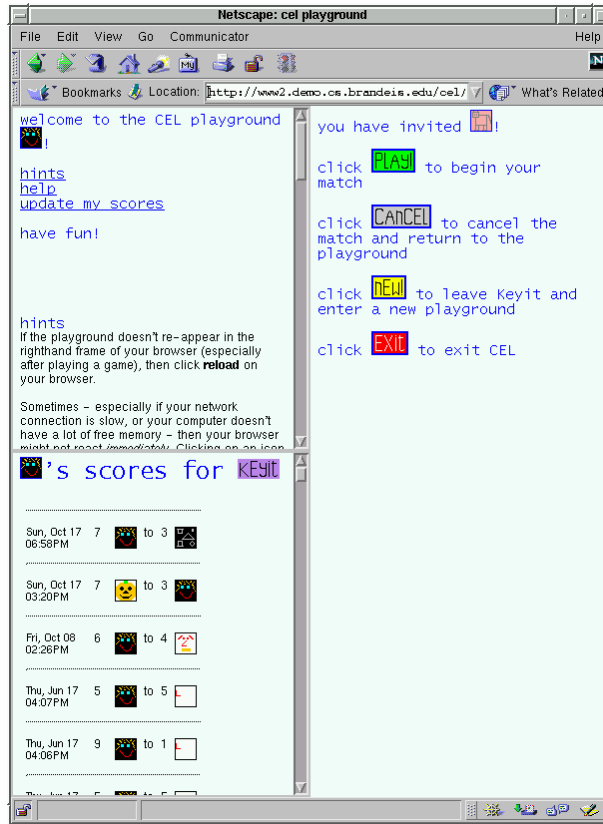


Figure 3.5: An invitation to play a match.

The match begins when the browser displays a game page, containing a Java applet that facilitates play. Both players participate according to the particular format of the selected game. When the match is over, each player is returned to his playground and is then free to engage in another match.

A map of the CEL site is shown in figure 3.6, indicating the relationship between the games menu page (figure 3.3), and the playground (figure 3.4) and match pages (for example, figures 3.8, 3.9, 3.10, 3.11 or 3.12). The unshaded boxes represent the static portion of the site. The shaded boxes illustrate playground and match pages, which are created dynamically for each activity in CEL, as users enter and exit playgrounds and initiate matches.

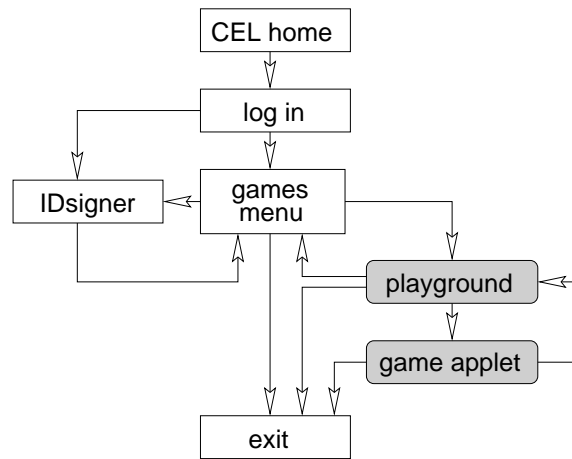


Figure 3.6: Site Map.

3.2 The IDsigner

Users create and modify their IDsigns using a pixel editing tool called the *IDsigner*¹ (see figure 3.7). Each IDsign is 20×20 pixels in size. First-time users must create an IDsign during the login process, before they can enter a play-

¹The IDsigner is similar in operation to the KidPix stamp editor.

ground. Participants may modify their IDsigns later, by selecting the IDsigner from the games menu.

Users are given a palette of 13 colors to choose from² and a straightforward point-and-click interface with which they can set the color of each of the 400 pixels. IDsigns are saved on our server, so when users return to CEL, the most recent version of their IDsign is loaded automatically.

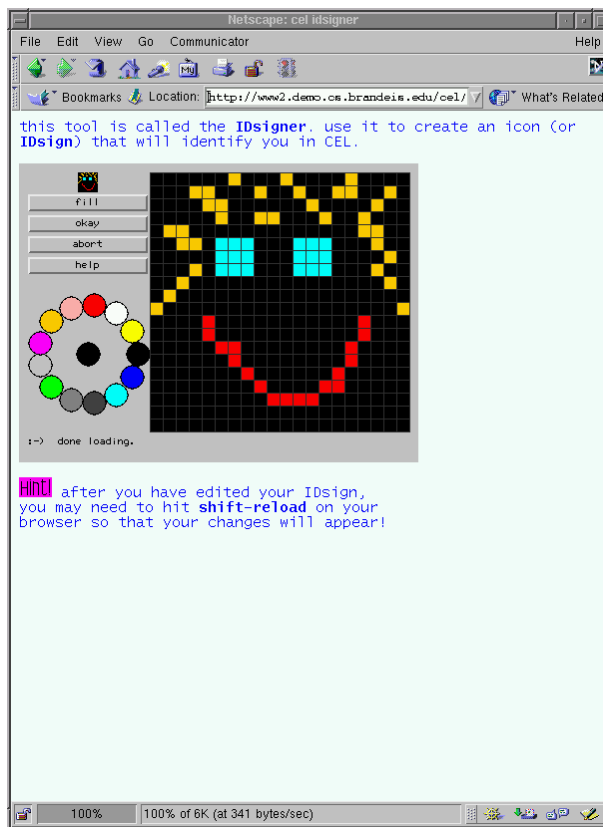


Figure 3.7: The IDsigner.

²The palette contains the standard set of web-safe colors defined in the Java class `java.awt.Color`.

3.3 Prototype Activities

For the prototype implementation of CEL, we built games that would let us demonstrate the flexibility of the system to host various types of activities. Based on our review of the interactive learning systems literature, we chose to focus on two-player games that are either competitive or collaborative and allow either synchronous (turn-taking) or asynchronous (real-time) interaction between players. The games were designed to be easily accessible by participants with computers that have limited memory and low network bandwidth. So we restricted the games to small footprint Java applets, which means they take less time to load on participants' computers and require little memory once loaded. These aspects are discussed further in chapter 4.

Currently, three word games, one math game, one construction activity and one spatial reasoning game have been built and tested. These are called *Keyit*, *Pickey*, *Monkey*, *Automath*, *Loois* and *Tron*. Each is described below. Two additional games are also in progress: *SpellebrityBee* and *Mathtree*.

3.3.1 Keyit

The keyboarding game called *Keyit*³ is pictured in figure 3.8.a, and a close-up is shown in figure 3.8.b. This is a competitive game in which participants are given ten words to type as fast as they can, with 100% accuracy. For each player, a timer begins when she enters the first letter of a word. Time is measured using

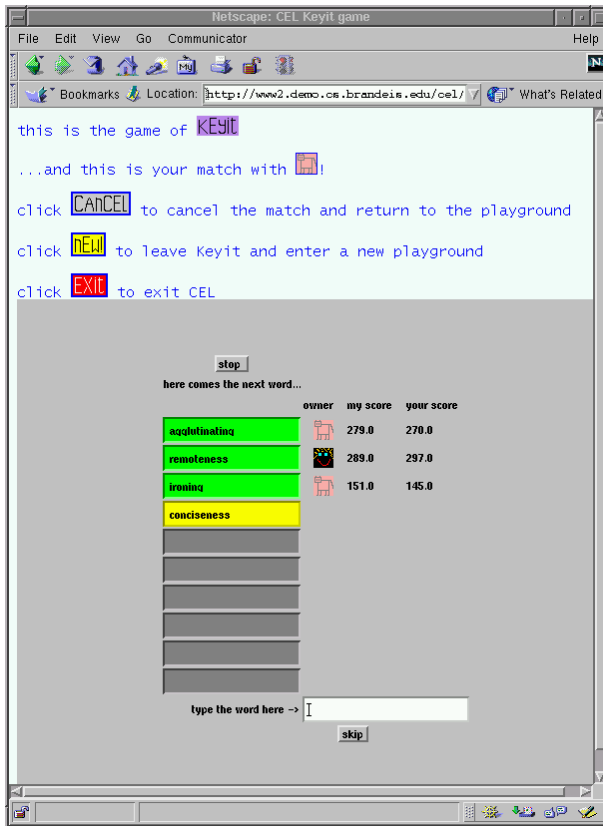
³Keyit was implemented in collaboration with Maccabee Levine and Travis Gebhardt.

the system clock on the client's computer, and a score is calculated, which is simply the time in hundredths of a second.




When the game page is first displayed, the column of words to type is empty. The user clicks on the "start" button, and the game begins. A word appears in the top row and is highlighted in yellow. The user attempts to type the word correctly, in the white text field provided at the bottom of the applet. By pressing the *Enter* key, the user signifies that she has finished typing the word. The system verifies the entry, and if it is correct, the word box turns green, and the user's time is displayed in the "my score" column. If the entry is incorrect, a message appears in the status message line (under the game button at the top of the applet): "*oops! try again.*" The user must correct the entry before being given the next word or she may opt to skip the word (by clicking on the "skip" button).

When the user's playmate completes the same word, her time appears in the "your score" column. Whoever has a lower score becomes the "owner" of the word, and her IDsign appears in the "owner" column. During the course of a game, feedback is provided to both players by filling in these columns as words are typed.

The match need not be synchronized. For example, a network link may be slow or one user may be interrupted. In this case, the system provides to each user whatever moves are available from their playmate.



(a) The game page.

	owner	my score	your score
extend		105.0	265.0
middle		200.0	86.0
par		25.0	24.0

(b) A close-up view.

Figure 3.8: The game of Keyit.

3.3.2 Pickey

The keyboarding game called Pickey⁴ is pictured in figure 3.9. This game is very similar to Keyit, except that users start with the full list of ten words, and they pick which ones they want to type.

The Pickey game board has two columns: the left column contains the list of words to be typed and the right column contains text fields where the player attempts to type each word. When the game page is first displayed, both columns are empty. The user clicks on the “start” button, the left column turns yellow and fills with words, and the game begins.

Play proceeds by the user selecting a word to type, clicking on the corresponding box in the “attempt” column (which turns white), typing the word and pressing the *Enter* key, to signify that she has finished typing the word. The system verifies the entry, and if it is correct, both boxes in the row turn green and the user’s time is displayed in the “my score” column. If the entry is incorrect, a message appears in the status message line (under the game button at the top of the applet): “*oops! try again.*” The user may correct her entry or pass on that word and go on to another one. The mechanisms for reporting scores and granting ownership of words are handled the same way as they are in Keyit.

⁴Pickey was implemented by John Abercrombie.

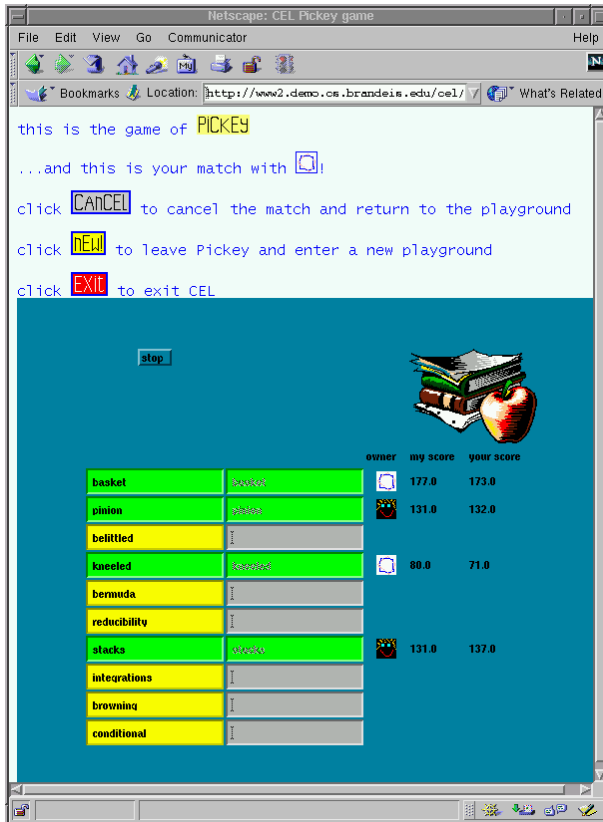


Figure 3.9: The game of Pickey.

3.3.3 Monkey

Monkey⁵ is a collaborative anagrams game in which players are given one long word and they work together to find as many smaller words as they can, using the letters from the long word. The long word is referred to as the “monkey’s word” and players “monkey around” with the letters in the monkey’s word to create new words, referred to as “sub-words”. When forming sub-words, letters may appear only as many times as they appear in the monkey’s word.

Here are some examples of valid sub-words for the monkey’s word “*alienation*”: *alien*, *nation*, *line* and *in*. An invalid sub-word would be “*teen*”, since the letter “*e*” appears only once in *alienation*, but twice in *teen*.

The Monkey game page also has a “start” button, which each player clicks on to begin his game. The monkey’s word is shown under the “start” button. Players enter sub-words in the text field at the bottom of the applet, ending with the *Enter* key. Each entry is validated by the system as follows. First, the entry is checked to make sure it is long enough (sub-words must be at least two letters long). Then it is checked to see if it has already been used in this game. Next, the system makes sure that the letters in the entry appear in the monkey’s word. Finally, the entry is verified by checking in a dictionary, to make sure it is a real word⁶. Valid sub-words are listed in the scrolling area which appears between

⁵Monkey was implemented in collaboration with John Abercrombie.

⁶Currently, we are using the standard dictionary that comes with Linux. We use this same dictionary as the database for all word games. We have filtered out any crude words by hand. This dictionary, although it contains almost 35,000 words, is incomplete. In future, we will

the monkey's word and the word entry field. Play continues until both players want to quit or until all valid sub-words have been found.

There is no concept of individual score in Monkey, because it is collaborative rather than competitive (like Keyit and Pickey). On each playground page for every game in CEL, users can see a record of all the matches they have played

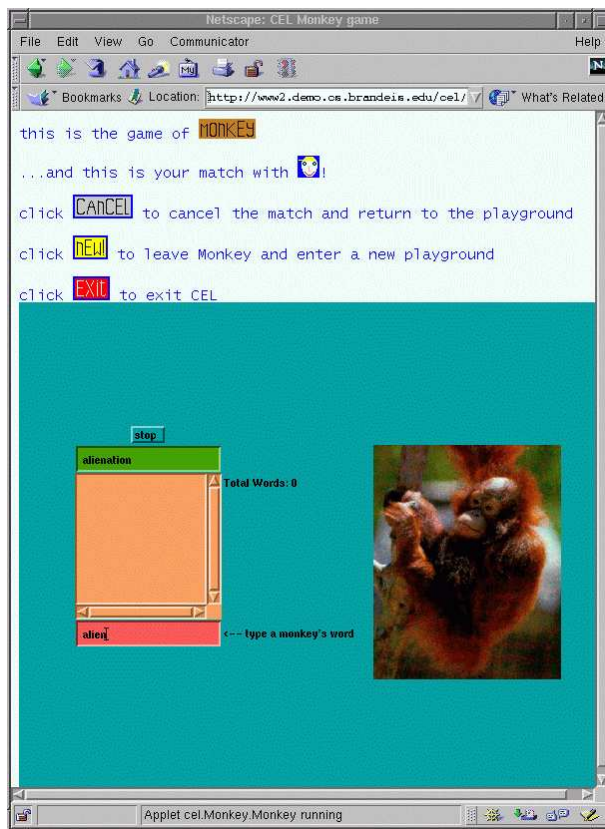


Figure 3.10: The game of Monkey.

replace it with a more comprehensive dictionary.

and some value associated with each match. For Keyit and Pickey, this is simply the number of words owned by each player owned at the end of the match. For Monkey, the total number of sub-words found by both players is shown.

3.3.4 Automath

The game of Automath⁷ is a competitive math game that follows the same structure as Keyit. However, instead of being given words to type, players are given equations to solve. All the equations are in the form: $a \langle operator \rangle b$. Valid operators are: + (addition), - (subtraction), * (multiplication), / (integer division) and \wedge (power).

Like Pickey, the Automath game board has two columns: one for the equations to be solved and one for the user's answers. But like Keyit, each row fills in one at a time. The game begins when each player clicks on her "start" button. The first equation turns yellow, and the user must solve the equation by typing the correct answer in the corresponding box to the right of the equation, ending with the *Enter* key. The user's entry is timed, and score is reported (the time, in hundredths of a second), in the "my score" and "your score" columns. The IDsign for "owner" of each problem — she who solves each equation correctly and more quickly — is shown in the same row as the problem. Players are allowed to skip an equation by clicking on the "skip" button.

⁷Automath was implemented by Elizabeth Sklar, based on the InOutMachine game [Sklar et al., 1998].



Figure 3.11: The game of Automath.

3.3.5 Loois

Loois⁸ is a collaborative construction game in which players work together to create structures out of building blocks. This is a turn-taking game. The player who initiates the match goes first. He selects a block from a bank of building blocks and uses his mouse to drag it onto the building area. When he releases his mouse, his move is sent to his partner. Every move is checked for structural integrity, using the Lego simulator built by Pablo Funes [Funes & Pollack, 1998a; Funes & Pollack, 1998b]. If any blocks are deemed unstable, they are highlighted in black in the building area.

When the players have finished constructing, they may print out a schematic, containing plans for building the structure they have designed on-line. This process promotes transference of information from the virtual world to the physical world and helps teach students about visualization, projection and dimensionality. For young children, learning how to assemble physical structures by following paper instructions is a valuable skill; because they have designed the structures themselves, they may more easily understand the relationship between the elements of the structure as they are represented on paper (and/or on a computer screen) and their physical instantiations. As well, this game serves to introduce children to the field of computer-aided design.

In an expanded version of this game, children will be able to work together to design entire cities. The buildings in the cities could be created by anyone

⁸The applet for Loois was built by Louis Lapat and Pablo Funes.

who plays the game, so participants in different places could contribute to a cooperative project. Children will be able to learn from their peers by observing structures built by others. A web page will show all the buildings in the city and their locations in relation to each other. Visitors to the web page will be able to print schematics of selected buildings. In this way, children on opposite sides of the world can reconstruct the same city, physically, in their own classrooms.

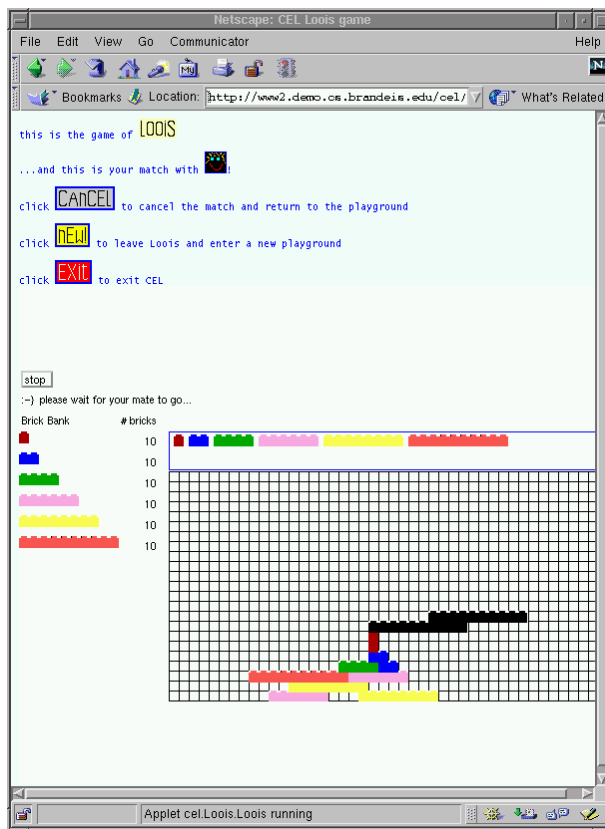


Figure 3.12: The game of Loois.

3.3.6 Tron

In earlier work [Funes et al., 1997; Funes et al., 1998], we built a Java version of the real-time video game Tron⁹ and released it on the Internet¹⁰ (see figure 3.13). Human visitors play against an evolving population of intelligent agents, controlled by genetic programs [Koza, 1992].

Tron became popular in the 1980's, when Disney released a film featuring futuristic motorcycles that run at constant speeds, making right angle turns and leaving solid wall trails behind them — until one crashes into a wall and dies. We abstract the motorcycles and represent them only by their trails. Two players — one human and one agent — start near the middle of the screen, heading in the same direction. Players may move past the edges of the screen and re-appear on the opposite side in a wrap-around, or *toroidal*, game arena. The size of the arena is 256×256 pixels. The game runs in simulated real-time (i.e., play is regulated by synchronized time steps).

Although Tron is not particularly educational, we placed a version of it in CEL because we wanted to demonstrate the ability of the CEL system to host a real-time, asynchronous activity. In the CEL version of Tron, participants play indirectly against each other by both competing against the same software agent.

⁹Tron was implemented in collaboration with Pablo Funes.

¹⁰<http://www.demo.cs.brandeis.edu/tron>

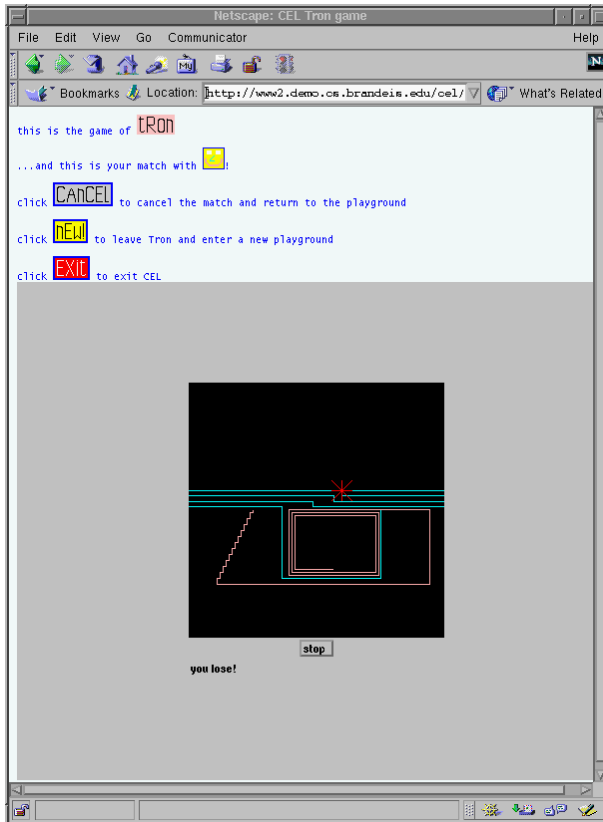


Figure 3.13: The game of Tron.

3.4 Summary

Table 3.1 contains the current CEL game set, highlighting the range of functionality amongst the games and the overall flexibility of the CEL system.

Table 3.1: Functionality of games in CEL.

	interaction	mode	domain
<i>Keyit</i>	real-time	competitive	keyboarding
<i>Pickey</i>	real-time	competitive	keyboarding
<i>Monkey</i>	real-time	collaborative	spelling, vocabulary
<i>SpellebrityBee</i>	turn-taking	competitive	spelling
<i>Automath</i>	real-time	competitive	math
<i>Mathtree</i>	real-time	collaborative	math
<i>Loois</i>	turn-taking	collaborative	construction
<i>Tron</i>	real-time	competitive	spatial reasoning

Each of the games described is implemented in Java and restricted to a small footprint applet, in order to provide easy access to participants with computers that have limited memory and slow network bandwidth. The pilot study described in chapter 6 demonstrates this accessibility of the system.

Keyit was the first game implemented. All the other games were built by undergraduate programmers by extending this model. Discussion of this extensible model is contained in chapter 4. Instructions for extending the model to create new games can be found in [Sklar, 2000].

Chapter 4

System Architecture

The CEL system employs a modular client-server architecture, as shown in figure 4.1. One central server maintains a dynamic database indicating who is logged into the system and which games they are playing. This server also acts as a message passer, sending and receiving commands that go between clients.

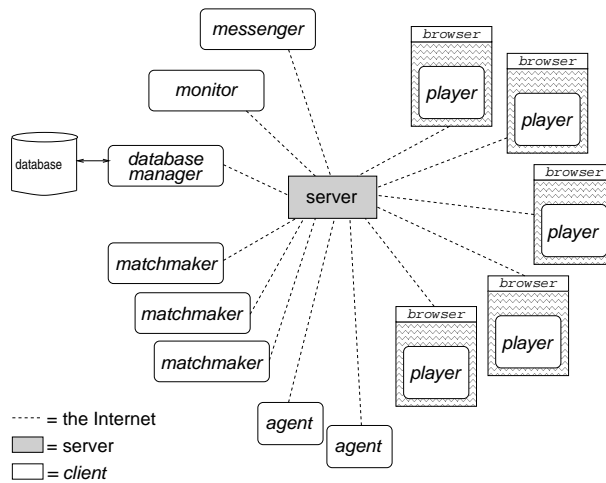


Figure 4.1: System Architecture.

There are six different types of clients in CEL: messenger, monitor, database manager, matchmaker, agent and player. The player client is designed to meet two fundamental needs: (1) to be practicable to anyone with Internet access and a web browser capable of running Java, and (2) to be usable by participants with limited network speed and low computer memory, as is the case for many school children. As such, we use small footprint Java applets for games and implement the playgrounds using CGI-bin programs that generate HTML and refresh periodically in order to update playgroup content.

The system is designed to be easily accessed by participants and easily extended by contributors (those adding their own games to CEL). Participants access CEL through the player client component, which runs inside their browsers. Section 4.7 highlights the characteristics of the player client that were built to meet the needs of school children. Contributors extend the matchmaker and agent components and the game portion of the player client component to implement their own activities in the system. Sections 4.5, 4.6 and 4.7, explain the details of each of these components, respectively.

This chapter describes each of the seven modules (one server and six types of client). Detailed software documentation can be found in [Sklar, 2000].

4.1 Server

The CEL server is a control component, having two primary functions: (1) to act as a central message processing facility, handling communication between all types of clients, and (2) to maintain a list of all the players who are currently logged into CEL and the status of each player. The server is written in Java, version 1.0.2. We use Java version 1.0.2 because it can run inside Netscape version 3, which is (currently) more widely used than later versions of Netscape — supporting CEL’s requirement for accessibility.¹

A note about terminology and formatting in this document: words that are Java keywords are highlighted in **this font**; words that are CEL keywords (e.g., CEL classes, variables and commands) are highlighted in **this font**.

The server interfaces with each of the six types of CEL clients (messenger, monitor, database manager, matchmaker, agent and player). The terminology can be somewhat confusing because while CEL players may be thought of as general “clients”, they are not the only type of client. And while matchmakers may also be referred to as “game servers”, they are really clients as well. The distinction comes from network communication phraseology: the server opens a `ServerSocket` and each type of client opens a `Socket` in order to send and receive messages to and from the server (see figure 4.2).

Commands are sent between the server and clients using the CEL command

¹Of course, we could use a later version of Java for our server and only restrict applet code to 1.0.2, but we decided it was simpler from a configuration management standpoint to use the same version for everything.

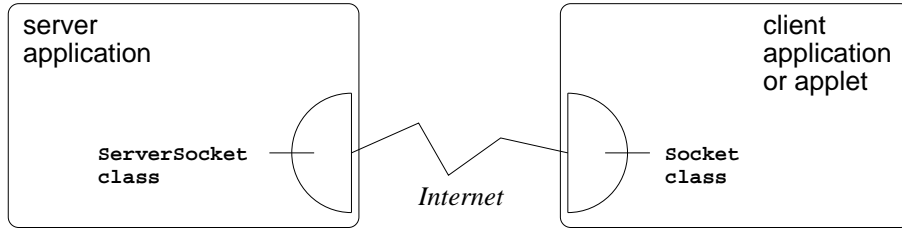


Figure 4.2: Sockets.

language (described in section 4.8). For example, messages are used to: log a client into and out of the system, register a player entering a playground or remove an exiting player, send a match invitation from one player to another, and pass game moves between players.

Figure 4.3 illustrates the **Server** and its relationship to each client application or applet. The components of the **Server** are shown above the solid grey line in the figure. The clients (shown below the solid grey line) are described in later sections of this chapter.

The **Server** extends the Java **Thread** class. It opens a **ServerSocket** on a specified port and listens for connections. When a new client makes a connection, the **Server** instantiates a **ServerClient Thread** to handle bi-directional communication with that client. The **Server** maintains a list of all active clients, i.e., a **Vector** of **ServerClients**.

As long as the socket connection with any **ServerClient** is alive, the server assumes that the client is running. When a client exits normally, it sends a **LOGOUT** command to the **Server** and closes the socket connection. Sometimes, the

Server will initiate the closing of a client, either because the **Server** is shutting down, or because it has received a command to kill a particular client, or because the socket connection has died, which typically happens when a client exits abnormally.

The **ServerCleanup** class monitors the status of every client connection, running periodically to check if any activity has occurred on each client's socket connection within a fixed time period. If no activity has occurred, then the **Server** sends a **PING** command to the client. The expected response is a **PONG** command, from the client back to the **Server**. If this is not received within a fixed time period, then the **Server** assumes that the client has exited abnormally and so the cleanup thread closes that client.

This process is necessary because we cannot ensure that clients (particularly players) will exit CEL cleanly. Players are instantiated in participants' browsers and if a user clicks away to another web site or closes his browser without logging out of CEL, then we have no way of knowing that the player has exited. So, in order to maintain the integrity of the active client list in the **Server**, we use the **ServerCleanup** thread. For example, this prevents the system from creating game matches that involve players who have left the system.

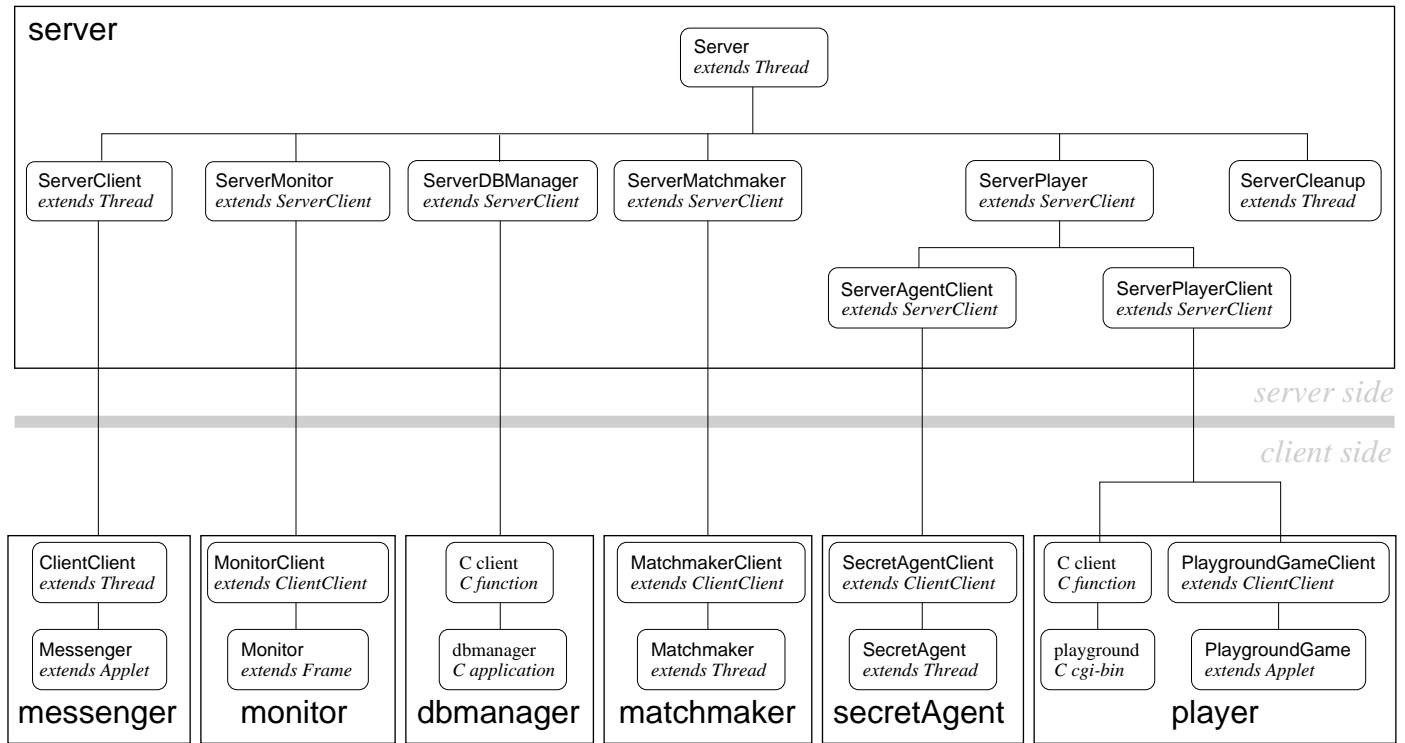


Figure 4.3: Server architecture, with overview of clients.

4.2 Messenger

The simplest client in CEL is the **Messenger** client. It is a Java application that provides a command-line interface for sending commands to the **Server**. It was built primarily as a development aid. The **Messenger** takes a message string in its command line and sends the message directly to the **Server**. The syntax of the message is the same as the CEL command language (see section 4.8).

4.3 Monitor

The **Monitor** is an expansion of the **Messenger**. It is also a Java application that provides a command-line interface for sending commands to the **Server**, but the **Monitor** also receives live feedback from the **Server**, reporting current status information on all active clients. The **Monitor** has a graphical front-end, which is pictured in figure 4.4. It can also run in a non-graphical mode, which is especially useful when testing CEL at a remote site where a graphics terminal is not available.

The **Monitor** can run on any networked computer, so it is a useful tool for contributors who are extending CEL.

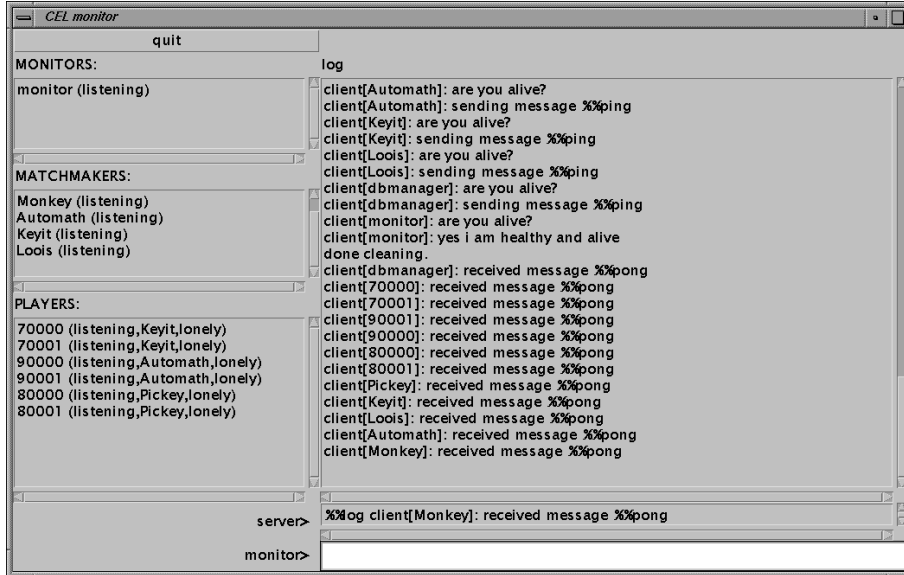


Figure 4.4: The Monitor.

4.4 Database Manager

The `dbmanager` is a C program that interfaces between the `Server` and the CEL databases. These include the student model component of the system as well as session logs.

The `dbmanager` runs as long as the `Server` is running. Whenever a player begins or ends a match, a message is sent from the `Server` to the `dbmanager`. The `dbmanager` parses the message and stores the relevant data in the appropriate CEL database table. Refer to chapter 5 for detailed information about the particulars of the CEL databases.

4.5 Matchmaker

For each game in CEL, there is one **Matchmaker**. This is a Java application that keeps track of all the users who are currently connected to its game. The **Matchmaker** is responsible for maintaining playgroups for each player, fetching game content at the start of a match, instantiating agents to play the game when playgroups are too small and verifying moves during game play.

The components of the **Matchmaker** are shown in figure 4.5. Some of the components are highlighted in grey, to indicate that there may be multiple instantiations of these classes. A new **MatchmakerGame** class is instantiated every time a match begins. This is essentially a data structure that stores information pertinent to individual matches. When a match ends, its corresponding **MatchmakerGame** is removed.

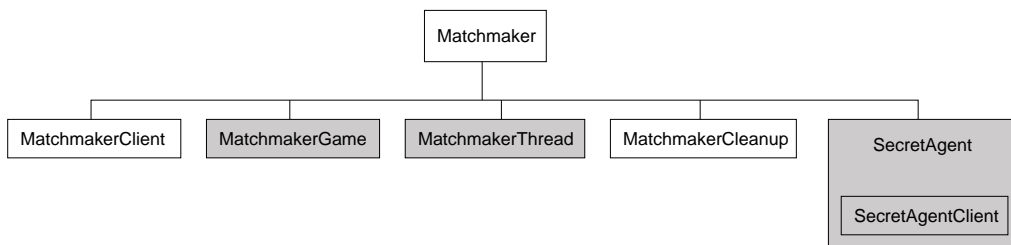


Figure 4.5: The **Matchmaker**.

The **MatchmakerThread** class can be used for one of two purposes: to fetch data for game content when a new match begins or to save data when a match is over. This thread is instantiated as either a **fetcher** or a **saver**. Both types invoke CGI-bin programs to interface, respectively, with the domain knowledge or the

student model component of the CEL databases.

The algorithm used to fetch game content can vary, in order to maintain system flexibility. For example, formative evaluation of a system might involve comparing different methods for selecting content for the same game, which can be done simply by employing different CGI-bin programs to fetch the game data.

The **SecretAgent** runs as a child of the **Matchmaker**. Once instantiated, a **SecretAgent** executes independently of its **Matchmaker** parent, although it dies when its parent dies. Secret agents can also be instantiated as programs and run autonomously, outside of a **Matchmaker**. This flexibility allows implementation of secret agents that have the ability to play multiple games. Detailed discussion of secret agents is found in section 4.6 and in chapter 7.

In order to implement their own activities, contributors extend the **Matchmaker**, **MatchmakerThread** and **SecretAgent** classes. For example, the Keyit matchmaker is enabled through the **KeyitMatchmaker**, **KeyitMatchmakerThread** and **KeyitSecretAgent** classes. Matchmaker applications can execute on contributors' local machines as well as on our site. Instructions for extending classes to create a matchmaker can be found in [Sklar, 2000].

4.6 Secret Agent

All the games in CEL are multi-player games. If not enough people are logged into a game playground, then it is useful to have software agents that can act as playmates. Otherwise, participants would be forced to wait until another human

logs in before being able to play any games. We refer to the software agents in CEL as **Secret Agents**, because the agents may be indistinguishable from human playmates.

Figure 4.6 illustrates standard architecture for a software agent [Russell & Norvig, 1995]. In CEL, the sensors and effectors are provided virtually by the **SecretAgentClient** class. Through the **Server**, the **SecretAgentClient** receives information about the state of the world and sends its actions to be effected.

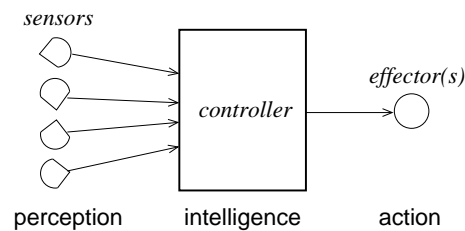


Figure 4.6: Typical software agent architecture.

The **SecretAgent** may perform three types of actions: system actions, playground actions and game actions. System actions refer to logging in and out of CEL. Playground actions refer to entering and exiting playgrounds and initiating challenges. Game actions refer to moves in a game.

The controller for a secret agent decides which action to take, given the input received from the **Server**. The flexibility of the CEL system makes it possible to design many types of controllers for secret agents. For the prototype version of CEL, we have defined two types of controllers. One is a simple reactive controller that does not initiate any challenges on the playground, and when playing a game,

all its moves are direct responses to its playmates moves, based on a rule that allows the playmate to win almost every match, by a small margin. The second type of controller is a neural network controller that is trained to emulate the behavior of humans who have visited CEL. Chapter 7 discusses the agents in more detail.

The **SecretAgent** class extends a Java **Thread**. It has one child: **SecretAgentClient**. Secret agents can be instantiated as part of a **Matchmaker** (see section 4.5). They can also be run as independent programs. This allows more flexibility, particularly by permitting complex agents that exhibit system actions like logging in and out of CEL at particular times and being able to play different games.

Contributors need to extend the **SecretAgent** class in order to implement an agent that can participate in a new activity. Refer to [Sklar, 2000] for instructions on how to extend this class.

4.7 Player

The CEL player client implements the user interface component of the system. It runs inside a web browser and has three elements: the menu, the playground and the game (see figure 4.7).

The menu and playground portions of the player client are implemented separately from the game portion. The following subsection details the development of the player module and adjustments made during formative assessment. The

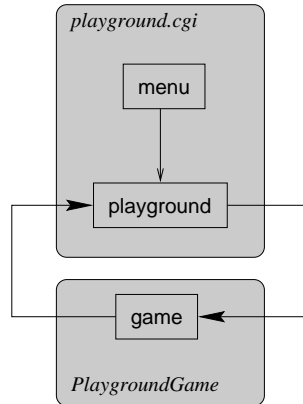


Figure 4.7: Overview of the player.

section concludes with a description of the final design.

4.7.1 Formative assessment

We performed formative assessment of the player client using the setup at a local primary school. The initial implementation of the player client was written entirely in Java. The playground was an applet, which opened a socket connection to the CEL server when it was launched and kept that connection open throughout a player’s entire session with CEL.

As illustrated in figure 4.8, the entire browser window was taken up with one playground applet. The background was an image, designed to look like the black-top in a school playground — white lines demarking hopscotch and a basketball court on a black background. Users’ IDsigns were inscribed in circles, whose color changed based on the state of each player. Players who were playing games were shown in green circles. Players who were sitting in the playground

were drawn in white circles. A user's own player was inscribed in a blue circle.

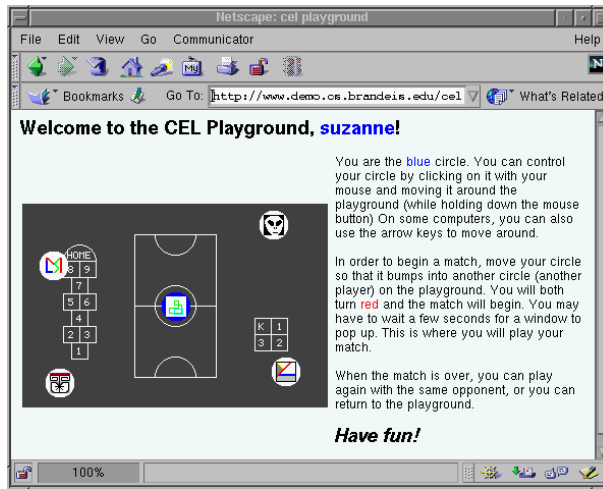


Figure 4.8: The CEL Playground, initial version.

The IDsigns were animated, moving around the playground in a fluid and dynamic manner. Each user controlled her own IDsign by clicking on it with her mouse. She could drag her IDsign and “bump” into that of a playmate; this action constituted an invitation to play a match. Playmates’ IDsigns moved in and out of the playground as users entered and exited the virtual space.

This design worked quite well in the laboratory, and children who tried this interface during one-to-one testing in our lab liked the style. Unfortunately, it proved too slow and cumbersome to be useful in a school setting. The amount of memory required for a large playground applet was too great for the computers at our test site, a local primary school. Opening a socket to the server and keeping it open continuously did not prove to be reliable. The idea of moving IDsigns around with a mouse, while appealing to the children, did not perform well in

practice, primarily due to limited memory.

We were fortunate to be able to work at a test site that is better equipped than most schools and may have faster access than many children do from home. But we wanted to make CEL accessible to school children across the U.S. and around the world, so the system had to perform to at least a median common denominator. If the performance was poor at our test site, we knew that it would not fare well in a typical school setting.

So, at first, we modified the design slightly, as shown in figure 4.9. Here the size of the applet was smaller (i.e., the amount of memory it used) because the image background was removed and the informational portion of the screen was done in HTML. Additionally, frames were introduced, so the left-hand portion of the screen, which was only HTML loaded up quickly and warned students to be patient while the right-hand frame (which contained the applet) loaded.

We were disappointed to find that these alterations did not resolve the problems at our test site. The issues with memory were compounded by socket connection breakdowns. When the applet initialized, it opened a connection to the CEL server and attempted to keep that connection open during a user's entire session with CEL. However, we found that these connections kept getting interrupted. We tried implementing a recovery process whereby interrupted connections would reconnect to the server. But this resulted in more memory problems, because the memory allocated for lost connections was not recovered well in the browser and so as more connections were made, less and less memory became available on the students' computers and eventually they would hang.

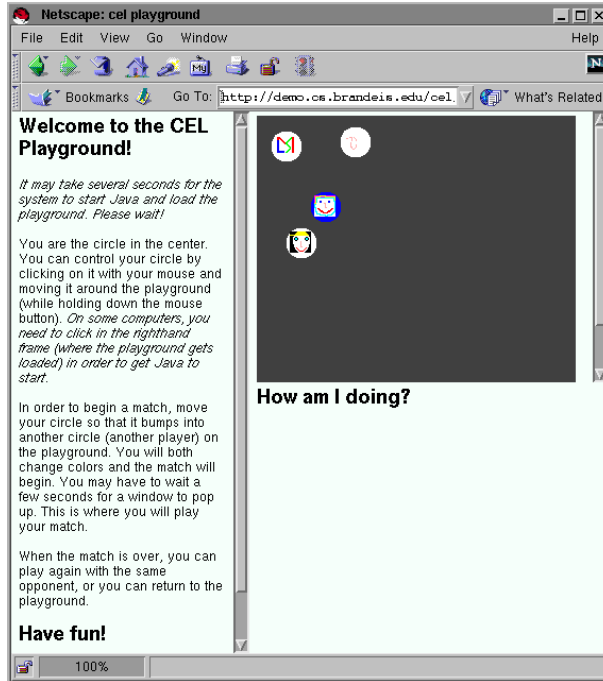


Figure 4.9: The CEL Playground, intermediate version.

As a result, we redesigned the player to use HTML and cue the browser to refresh the playground page periodically, asking the server for playgroup updates. This means that the player does not need to maintain a single long-term socket connection to the server. Conceptually, the playground is still an ongoing process that maintains a connection to the server so that the player can receive updates concerning who his current playmates are, as others enter and exit the site. In reality, the playground sends a refresh command to the browser telling it to update the playground every 5 seconds, thereby simulating a real-time connection to the server. Using this type of polling mechanism makes the system more accessible, because it accommodates clients with slow network connections and

low memory computers.

The product of the redesign is a program called `playground.cgi`. At first, this was a shell script which invoked a Java application that connected to the `Server`, received an updated list of playmates and output HTML to draw the playground in the user's browser. After a few trials, we found that this method was also unacceptable. Every time the script was called, it started a Java virtual machine on our host computer in order to execute the Java application. This is bad because each Java virtual machine takes a while to start up, uses a lot of system resources while it runs and takes a while to close down and release the resources it was using — typically longer than the 5-second refresh period of each playground. When more than about 5 people were logged into CEL, our host machine was swamped. One player could have several `playground.cgi`'s running on our host because their browser would send a refresh command and start up a new one before the previous one(s) had completely exited the system.

4.7.2 Final design

The final `playground.cgi` is written in C and implements the menu and playground portion of the player client. This version runs in less than one second, and it works beautifully in practice. Although we gave up the fun gained from moving one's IDsign around in an animated environment, we are still able to emulate the dynamic nature of the environment and have ended up with a much more reliable and accessible product.

The game is a small-footprint Java applet, started when the `playground.cgi`

outputs HTML containing an `APPLET` command. The applet is built on a class called `PlaygroundGame`. When a `PlaygroundGame` applet initializes, it creates a child called `PlaygroundGameClient` that opens a socket to the `Server` and keeps that connection open as long as the game is in progress. When the game is over, the socket is closed and the applet invokes `playground.cgi` again, to fetch an updated playgroup from the `Server` and return to the state of waiting for a match. Again, accessibility is important, so the size of the applets are kept to a minimum, in order to lessen the memory requirements of clients' computers.

The user interfaces for individual games are enabled by extending the `PlaygroundGame` class. All of the games described in chapter 3 are built around this model. Contributors extend the `PlaygroundGame` class to implement the user interface for their own activities.

4.8 CEL Message Language

All communication between the `Server` and its clients is facilitated by the CEL Message Language. There are five classes of messages that are sent:

- I – commands between the server and any type of client
- II – commands between the server and messenger and monitor clients
- III – commands between the server and dbmanager client
- IV – commands between the server and matchmaker clients
- V – commands between the server and player and agent clients

Tables 4.1 through 4.5 list each set of commands, respectively, and describe their actions.

The message language was designed to support the extensibility requirements of the system. The separation between command classes protect the system; contributors who create matchmakers, players and agents cannot send destructive commands to the **Server**. For example, a matchmaker cannot effect a **KILL** command because it will not be recognized as a valid matchmaker command in the server.

The commands in classes III, IV and V were designed for flexibility, keeping short and simple the amount and type of communication that flows between the server and dbmanager, matchmakers, players and agents. The command set can support a variety of activities, as outlined in chapter 3.

Table 4.1: Class I: commands between server and any type of client.

<i>command</i>	<i>from server to client:</i>	<i>from client to server:</i>
LOGIN		client logs into CEL
LOGOUT		client logs out of CEL
PING	server checks socket connection	
PONG		client verifies socket connection
GETTIMEOUT	server reports timeout value for client	client receives value of timeout
SETTIMEOUT		server sets value of timeout
SEND	server sends a message to a client	client sends a message to another client, via server
ERROR		client reports that an error has occurred
SHUTDOWN	client shuts down	server shuts down (<i>messenger and monitor clients only</i>)

Table 4.2: Class II: commands between server and messenger/monitor clients.

<i>command</i>	<i>from server to client:</i>	<i>from client to server:</i>
LOG		server writes a message to the log file
FLUSH		server flushes the log file
GETLOGINT	server reports value of log interval used during cleanup	client receives value of log interval used during cleanup
SETLOGINT		server sets value of log interval used during cleanup
GETCLNUPINT	server reports value of cleanup interval	client receives value of cleanup interval
SETCLNUPINT		server sets value of cleanup interval
CLEAN		server forces a cleanup to occur
WHO	server reports list of active clients	client receives list of active clients
KILL		server kills specified client

Table 4.3: Class III: Commands between server and dbmanager clients.

<i>command</i>	<i>from server to client:</i>	<i>from client to server:</i>
ENTER	server sends name of player entering a playground and time of entry	
EXIT	server sends name of player exiting a playground and time of exit	
RESULTS	server sends results of a match	

Table 4.4: Class IV: Commands between server and matchmaker clients.

<i>command</i>	<i>from server to client:</i>	<i>from client to server:</i>
GAME	server sends request for matchmaker to fetch game data	
DATA		matchmaker sends game data to server
RESULTS	server sends results of a match	

Table 4.5: Class V: Commands between server and player/agent clients.

<i>command</i>	<i>from server to player:</i>	<i>from player to server:</i>
ENTER		player enters a playground
EXIT		player exits a playground
STATE	server gets current state (list of active playmates)	player receives current state, with which to update playground
ASK		player requests match with specified playmate
GAME	match with requested playmate is accepted and server has requested game data (agent) from matchmaker	player has started game applet and requests game data (agent) from server
DATA	server sends game data (after data has arrived from matchmaker)	
MOVE	server passes move along to opponent	player sends its move to server
RESULTS		player sends results to server (and server passes them along to matchmaker)
REJECT	server rejects match with requested playmate	
ABORT	client aborts match, returns to lonely state	server forfeits match for client and returns client to lonely state
ERROR	client exits	server removes client

4.9 Player States

Both the **Server** and the **Player** keep track of a player's state. At any time, each player is in one of five states:

1. **ENTERING** — a new player is entering a playground
2. **LONELY** — a player is in the playground and is free to initiate a match or be invited by another player to engage in a match
3. **PREGAME** — a player has initiated a match or has been invited to play and is waiting for game data to arrive from the server
4. **GAME** — a player is playing a match
5. **EXITING** — a player is exiting a playground

Figure 4.10 summarizes the state transitions for players inside the **Server**, showing the normal flow between the five states.

All changes to the state are initiated by the **Player**. This is important. Sometimes the **Server** and a player can become out of synch. It is most critical for the actions in the player to seem logical to the human who is operating the player. So, all state changes are initiated by the player, and all discrepancies are resolved in favor of the player.

There are several ways in which the player and the **Server** can become inconsistent. Since the player is enabled inside a browser, a user could click on a navigation button to change the page and visit CEL pages out of sequence. For example, in the middle of a game, the user could use the “back” button

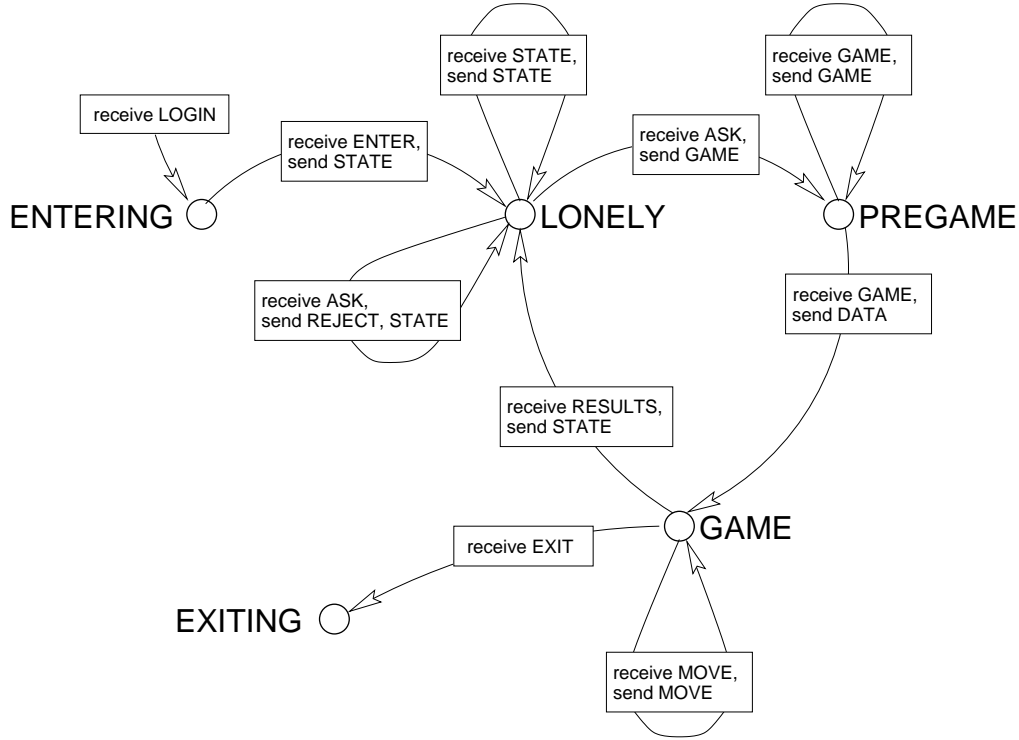


Figure 4.10: Player state diagram.

in the browser to visit a playground that is several pages prior in the history list. In this case, the **Server** thinks that the user is in a **GAME** state, but upon receiving a **STATE** command from the player client, the **Server** should respond intelligently. *The player is always considered right* — so the **Server** registers the player as having aborted the match that the server was tracking, and then adjusts accordingly.

If response time is slow, users (especially children!) grow impatient. Sometimes they click on a **CEL** button multiple times or on the browser’s reload button, while waiting for a response from the **Server**, which may result in an

unexpected sequence of commands being sent to the **Server**. The mechanism described here is prepared to handle these types of discrepancies, which works to support the accessibility of the system, as CEL can operate robustly, responding reasonably to a variety of user behaviors.

Chapter 5

Data in CEL

This chapter describes the data components of the CEL system, which include domain knowledge as well as data collected by the system as it runs. The domain knowledge is defined to be the information that the students are acquiring; in CEL, this is the content of games. We focus on three categories of on-line data collection in CEL (system products, session logs and survey results), demonstrating that CEL gathers the types of data required to support the kinds of activities common to the ILS field (see chapter 2).

5.1 Domain knowledge

Two types of domain knowledge databases have been defined for use by the CEL prototype activities:

1. a *words* database, and
2. an *arithmetic* database.

Keyit, Pickey and Monkey access the words database. Automath uses the arithmetic database.

The manner in which this data is stored and accessed is completely dependent on the learning activities that use the data. As indicated in chapter 2, the purpose of defining a domain database is to be able to identify elements within that database, in order to track a student's progress and (in CEL) to define game content. Some activities are designed to facilitate acquisition of a straightforward database of facts, e.g., multiplication tables, states and capitals, foreign language vocabulary. Database definition is easy in these cases because the domain can be broken down into individual elements and a one-to-one correspondence can be found between domain elements and elements of game content. For example, " 5×6 " can be defined as one element in a multiplication table database and this same equation can also be one (or part of a) problem to solve in a mathematics game. In the student model, the number of times a student has been asked to solve " 5×6 " can be tracked along with the number of times she has gotten the right answer, and then a simple numeric calculation can provide an indication of how well she has acquired that multiplication fact (e.g., percentage of correct

responses).

Other domains are much harder to define. In the construction game *Loois* (section 3.3.5), it is difficult to enumerate the concepts that students should be acquiring. Here, the elements of game content are numbers and sizes of building blocks. But the intent is for students to acquire abstract skills such as elementary laws of physics, an understanding of torque, intuition about gravity, insights into structural integrity and experience translating ideas from simulation to reality. Keeping track of skill acquisition here is much more complicated and is a research topic addressed by many working in intelligent tutoring systems. In future work with CEL, we plan to collaborate with others working in these areas to develop stronger methods for tracking student progress in complex domains.

The domain knowledge data sets that are used in the prototype version of CEL are described in the remainder of this section. The methodology defined should be taken as a suggestion for other possible domains and learning activities, but is not a requirement of the system.

5.1.1 *Words database*

The content of word games in CEL is selected from a database of approximately 35,000 words. Every word is characterized by a set of seven features:

1. word length,
2. Scrabble¹ score
3. keyboarding level,
4. number of vowels,
5. number of consonants,
6. number of 2-character consonant clusters and
7. number of 3-character consonant clusters.

The definition of Scrabble score is shown in table 5.1. For each word in the dictionary, the word's score is computed by adding together the scores for each character in the word. For example, the score for "millennium" is $3 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 3 = 13$.

The definition of keyboarding level is shown in table 5.2. There are several standards which define an order for introducing keys to students learning typing [Rowe et al., 1967; Duncan et al., 1990; Typodrome, 1997]; the latter was chosen arbitrarily. Each word in the dictionary is assigned a group number equal to the highest keyboarding level of any of the letters in that word.

¹Scrabble is a word game that was invented in the U.S. by Alfred M. Butts in 1948 [Hasbro, 1999]. It is a board game in which players take turns making interconnecting words by placing letter tiles on a grid in crossword puzzle fashion. Each letter is assigned a fixed value, calculated according to its frequency of usage in everyday American English. Players receive a score for each word they place, calculated by summing the values for each letter in the word. In the 1950's and 60's, Scrabble gained popularity and spread to Canada, Great Britain and Australia. Today, the game is known all over the world.

Table 5.1: Scrabble scores.

character	score	character	score
A	1	N	1
B	3	O	1
C	3	P	3
D	2	Q	10
E	1	R	1
F	4	S	1
G	2	T	1
H	4	U	1
I	1	V	4
J	8	W	4
K	5	X	8
L	1	Y	4
M	3	Z	10

Table 5.2: Keyboarding levels.

level	keys introduced
1	a s d f j k l
2	e u
3	r i
4	g o
5	t h
6	w y
7	q p
8	c v
9	b m
10	x n
11	z

Figure 5.1 shows an example of the feature vector definition for the word “blue”. The reason for defining features for words is to provide a basis for selecting words as the content for word games. Some algorithms for selecting game content require that the domain be divided into a multi-dimensional feature space, so that data elements can be ordered in some fashion. For example, this allows the domain space to be partitioned into “easy” and “hard” segments.

Not all features will be relevant for all applications. For example, keyboarding level will not be relevant for spelling games. Experimentation may highlight which features are relevant for which applications (see chapter 8).

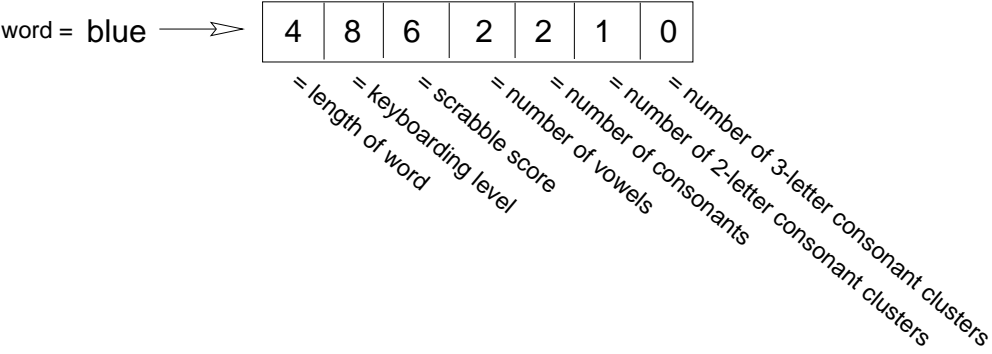


Figure 5.1: Sample word with feature vector.

5.1.2 Arithmetic database

For math games, the same level of traceability provided for word games is desirable — so that a student’s progress with a certain problem (or type of problem) can be tracked. Defining an element of knowledge in the arithmetic domain is more complicated than the method used for the words database. Is a formula the same as an element, e.g., “ $4 + 15$ ”? Or is the answer to a problem the same as an element, e.g., “19”? The answer taken by itself seems too vague, but if it is a formula, then should “ $4 + 15$ ” be the same element as “ $15 + 4$ ”? What happens with complex problems, like “ $4 + 15 - (8 * 7)^2$ ”? Is this one element or a combination of several?

Automath is the only currently operational math game, so the definitions used for this game will be presented here. An element is defined as follows for **Automath**:

$\langle \text{sign}_x \rangle \langle x \rangle \langle \text{op} \rangle \langle \text{sign}_y \rangle \langle y \rangle$

where:

$\langle \text{sign}_x \rangle, \langle \text{sign}_y \rangle$	=	positive (= 0) or negative (= 1)
$\langle x \rangle, \langle y \rangle$	=	0...1000
$\langle \text{op} \rangle$	=	addition (= 0), subtraction (= 1), multiplication (= 2), division (= 3), power (= 4)

To get a numeric identifier out of this, a binary string is constructed, by concatenating the numeric values for each of the five fields shown above.

Here is an example. The equation “2 – 5” is represented as follows:

<y>	<sign_y>	<x>	<sign_x>	<op>
5	+	2	+	-
000000101	0	000000100	0	001

Concatenating the values in the third row results in the binary number:

00000010100000001000001₂

which equals

163873₁₀,

and hence 163873 is the identifier for “2 – 5”.

This scheme does not allow for more complex formulae. For example, in the game of *Mathtree* (an arithmetic game that is currently being built), it is possible to have equations such as “4 + 15 – (8 * 7)²”. It is desirable to use the same domain element definition for all math games (as with all word games), so that a user’s performance can be described with respect to the knowledge base and the task (i.e., a word and how to spell it) rather than the specific game. As more activities in mathematics domains are implemented, it may be necessary to alter our definition.

5.2 Data products

There are three categories of data collected in CEL: system products, session logs and survey results. The system products consist of student models and results of games. Some of the data is stored in a Postgres SQL database. Some data is stored in text and binary files. For the dynamic tasks that occur during system operation, access to the data must be quick, so standard disk files (text and binary) are employed. For post-operations analysis, it is more important that data be easily queried, so Postgres database tables are also employed. This section describes each type of data collected in CEL.

5.2.1 Student model

The student model has three components: demographics, behavior and performance data. Each is detailed in the following pages.

Demographics

User demographic data is stored in two tables in the Postgres database: **users** and **demographics**. Data is inserted into these tables when users log into CEL for the first time.

The **users** table contains the following information:

- username** – a text value provided by users when logging into CEL for the first time.
- password** – a text value provided by users when logging into CEL for the first time. The **password** is encrypted using the C function `crypt()`.
- userid** – a numeric value assigned by selecting the maximum **userid** from the table and adding 1 to it.
- consent** – a boolean value which is **true** when the user has clicked **okay** on a shrink-wrap consent form page, giving us permission to collect data on users' activities.

The **demographics** table contains the following information:

- userid** – a numeric value that is the same as in the **users** table and can be used to cross-reference the two tables.
- gender** – a character value that specifies the user's gender (either "*m*" (male) or "*f*" (female)).
- age** – an integer value specifying the age of the user.
- location** – a text value that contains the country or state (if the country is U.S.) where the user is connecting from.
- address** – a text value containing the user's Internet (IP) address.
- language** – a text field that specifies the user's native language.

Behavior

Behavior data is stored in the **behavior** table in the Postgres database. This table is written to by the **dbmanager**. Every time a user enters or exits a playground, initiates, finishes or forfeits a match, an entry is inserted into this table. Thus the data in this table can be accessed in order to study users' behavior. For example, it is useful for training secret agents to emulate the playground behavior of humans.

The **behavior** table contains the following information:

- userid** – a numeric value that is the same as in the **users** table and can be used to cross-reference the two tables.
- action** – a text value that is one of the following:
 - ENTER (when a player enters a playground)
 - EXIT (when a player exits a playground)
 - ASK (when a player initiates a match)
 - RESULTS (when a player finishes a match)
 - FORFEIT (when a player cancels a match)
- game** – a text value indicating the name of the game in which the action has occurred (e.g., “Pickey”).
- timestamp** – a numeric value specifying the time when the action occurred.

5.2.2 Performance

For each user, performance data is maintained both in a **rates** file and in the Postgres database. The **rates** files are updated and accessed by the **Matchmaker** applications. The database tables are updated by the **dbmanager** and accessed for post-operations analysis.

Rates files

The **rates** files are stored in binary form. One file is stored per user. Some rates files may be shared by multiple games, as is the case for Keyit and Pickey. Monkey has a separate rates file, as does Automath. The content of each file is:

$$\langle n_r \rangle \langle r_0 \rangle \langle r_1 \rangle \dots \langle r_{n_r-1} \rangle \langle n_g \rangle \langle g_0 \rangle \langle g_1 \rangle \dots \langle g_{n_g-1} \rangle \langle avg \rangle$$

where:

- n_r = integer, number of r_i records
- r_i = a structure, of type RATE
- n_g = integer, number of g_i values
- g_i = integer, list of indices of domain elements in the last “generation” — i.e., the last problem set this user was given
- avg = double, a game-dependent player performance average

This format is the same for all rates files. The definition of the RATE structure varies from one activity to another (see figures 5.2 and 5.3).

```
typedef struct {
    int    elemid;
    int    count;
    double sum;
} RATE;
```

Figure 5.2: RATE definition for Keyit, Pickey and Automath.

$n_g = 10$ and $avg =$ the average score ($sum/count$) for all r_i records.

For each domain element: the number of times the user has encountered that element (**count**) and the total of all scores for all encounters (**sum**); thus an average rate for that element = $sum/count$.

```
typedef struct {
    int elemid;
    int count;
    int sum;
    int found;
} RATE;
```

Figure 5.3: RATE definition for Monkey.

$n_g = 1$ and $avg =$ the average ($sum/count$) for all r_i records.

For each domain element: the total number of times this word was the monkey’s word (**count**) and the total number of words found when this word was the monkey’s word (**sum**); thus an average number of words found when this word was the monkey’s word = $sum/count$ and the number of times the user has found this word (**found**) inside the monkey’s word (see section 3.3.3).

Rates tables

The **rates** tables are stored in the Postgres database. There are three inter-related tables, generally one set per activity, though games can share **rates** tables². The tables have the name of the game which they belong to appended to the table name:

1. *rate_game* (e.g., *rate_Keyit*),
2. *rate_x_game* (e.g., *rate_x_Keyit*), and
3. *gen_x_game* (e.g., *gen_x_Keyit*).

The **userid** field is common to all three tables and is used to join them. It is the primary key for all the tables.

The *rate_game* table is the master table. It contains one entry for each user (**userid**), which is akin to one **rates** file per user. The *rate_game* table contains the following information:

- | | | |
|----------------|---|--|
| userid | – | a numeric value that is the same as in the users table and can be used to cross-reference the two tables. |
| average | – | a real value akin to the avg field in the rates files. |
| n_rate | – | an integer value akin to the n_r field in the rates files. |
| n_gen | – | an integer value akin to the n_g field in the rates files. |

The *rate_x_game* table contains one entry for each domain element that a user has been exposed to. It is akin to the r_i array in the **rates** files. For every record in the *rate_game* table, there are *rate_game.n_rate* entries in the *rate_x_game* table, all with the same **userid**.

²Pickey shares Keyit's tables.

For example, the `rate_Keyit` table contains the following information:

- `userid` – a numeric value that is the same as in the `users` table and can be used to cross-reference the two tables.
- `elemid` – an index pointing to a word in the words database.
- `count` – an integer value indicating the number of times this user has been exposed to this word (`elemid`).
- `sum` – a real value containing the sum of the user's scores with this word for all `count` times seeing this word.

The `gen_x_game` table contains one entry for each domain element in the last problem set the user attempted. It is akin to the `gi` array in the `rates` files. For every record in the `rate_game` table, there are `rate_game.n_gen` entries in the `gen_x_game` table, all with the same `userid`.

The `gen_x_game` table contains the following information:

- `userid` – a numeric value that is the same as in the `users` table and can be used to cross-reference the two tables.
- `elemid` – an index pointing to an element in the domain database.

5.2.3 Match Results

For each match played, results data is maintained in both a journal file and in the Postgres database. The journal file is updated by the `Matchmaker` applications and is accessed by the programs that report to users a record of their activities with each game. The database tables are updated by the `dbmanager` and are accessed for post-operations analysis.

The journal files are stored in text form. One file is stored per game. The format of the journal files is as follows:

<timestamp><client><userid><match><result>

where:

<timestamp> = number of seconds since midnight on 01-Jan-1970
<client> = the IP address of the player client
<userid> = the player's userid number
<match> = four-field entity that uniquely identifies every match
<result> = result of the match, which varies for each game

The four-field <match> is defined as:

<timestamp><game><userid1><userid2>

where:

<timestamp> = number of milliseconds since midnight on 01-Jan-1970
<game> = name of game, e.g., **Keyit**
<userid1> = userid number of player 1
<userid2> = userid number of player 2

When a player finishes a match, a record is written to the journal file for that player, with that player's userid number as the third field in the record. This means that for matches where both players finish normally, there will be two records in the journal file for that match: one with player 1's **userid** in the third column and one with player 2's **userid** in the third column.

Note that the <match> field always lists the players in the same order, no matter whose results are contained in the record. Player 1 is defined to be the player who initiated the match. Player 2 is the player who accepted. If the match is between a human and a software agent, then the agent is always player 2.

5.2.4 Survey Results

Whenever a user exits a playground, she has the option of completing a quick on-line survey and answering two questions:

1. how hard was the match?
2. how much did you enjoy the match?

Both questions are answered on a scale of 1-10. For the first question, 1 is defined to be “easy” and 10 is defined to be “hard”. For the second question, 1 is defined to be “boring” and 10 is defined to be “exciting!”.

Answers to this survey are stored in the Postgres database in the **opinion** table. Records are inserted into the table by a CGI-bin program which displays an HTML form with the two questions on it and processes users’ responses. The table is accessed for post-operations analysis.

The **opinion** table contains the following information:

userid	–	a numeric value that is the same as in the users table and can be used to cross-reference the two tables.
timestamp	–	a date value containing the time the survey was completed.
game	–	a text value that specifies the name of the game played.
funness	–	an integer value between 1 and 10.
hardness	–	an integer value between 1 and 10.

5.2.5 System Logs

The system log file written in CEL maintains a chronological record of everything that happens while the system runs. This includes diagnostic information, which is kept for a month and then flushed.

The content of each log file is a record in the following format:

`<timestamp><message>`

The `<timestamp>` format is fixed (time in milliseconds since midnight, 01-Jan-1970). The `<message>` format varies. [Sklar, 2000] contains detailed information on the content of the system logs. Most messages are formatted as follows:

`client[<client>] : received message <message>`

`client[<client>] : sent message <message>`

These tags indicate which client was involved in the communication.

Some examples are shown below (the timestamps were removed):

```
client[174]: received message %%pong
client[Pickey]: received message %%pong

client[null]: received message %%login 3 4 929645102
client[3]: received message %%ask Keyit 4
client[3]: sending message %%game 929645104412 Keyit 3 4

client[Keyit]: sending message %%game 929645104412 Keyit 3 4
client[Keyit]: received message %%agent 929645104412 Keyit %%3 4 007 -31 10...
...11766 -1 embeds 9169 -1 curiousest 10667 -1 disinterestedness 22151 -1...

client[3]: received message %%move 929645104412 Keyit 3 4 3 11766 107.0 embeds
client[Keyit]: sending message %%move 929645104412 Keyit 3 4 3 11766 107.0 embeds
client[Keyit]: received message %%move 929645104412 Keyit 3 4 3 11766 107.0
client[3]: sending message %%move 929645104412 Keyit 3 4 3 11766 107.0
client[4]: sending message %%move 929645104412 Keyit 3 4 3 11766 107.0
client[3]: received message %%move 929645104412 Keyit 3 4 3 9169 225.0 curiousest
```

5.3 Summary

The data products collected in CEL may be accessed for many purposes:

- Student models may be accessed to select game content tailored to the needs of individual users. This is a dynamic task that occurs while the system is running and must contain current information that is consulted before every game a user plays.
- Student models may be used to define playgroup content. This is also a dynamic task that occurs while the system is running. Playgroups are updated after every game finishes and each time a new player enters or exits a playground.
- Match results are reported to users when requested. Playground pages show lists of all the matches a player has engaged in and the results. These lists update when a playground page is first loaded or dynamically if a user requests an update.
- User behavior and performance data may be used to train secret agents. See chapter 7 for an example.
- All types of data may be accessed for analysis, external to system operation.
- System logs are used to debug problems with the system.

The next chapter contains examples of the types of analysis that are typically performed in interactive learning systems, using the data products described

here. We used many software tools (shell scripts, C programs, Matlab) in order to analyse the data. Future work involves building a high-level set of analysis tools that teachers can use to produce the same types of tables and graphs shown here.

Chapter 6

Pilot Testing

During the first half of 1999, we conducted pilot testing with CEL. Forty-four fourth and fifth grade students from a local public primary school participated. Table 6 shows the breakdown of students, according to grade level and gender. All participants had signed parental permission. Any names of participants mentioned in this thesis are fictitious, in order to protect the privacy of the children.

Table 6.1: Breakdown of students.

grade	gender	number of students
4	male	10
4	female	12
5	male	9
5	female	13

The pilot testing took place in a computer lab in the primary school, where 15 iMac computers are connected to the Internet via a high-speed link. All children in this school make regular use of this lab for various activities, both

on- and off-line; thus the setting was familiar and comfortable. The children visited the computer lab for about an hour once a week and “did CEL” (under the author’s supervision). In most sessions, due to scheduling constraints, only children from the same grade were in the lab at the same time.

All the children did not have equal amounts of time to spend in the computer lab. Generally, the classroom teachers decided who would be allowed to go to the lab, typically based on other activities that were going on inside the classroom and whether each child had other classwork that needed to be completed first. With the fourth grade class, we usually took one group of students to the lab and they stayed for the entire session. With the fifth grade class, an initial group of children would come to the lab and then others would arrive and switch places with their classmates.

For all children, initial sessions required guidance either from the author or from another child who was already familiar with CEL. However, within 5 minutes or less, every child was able to get around in the community with ease. On-line help was available, although it was limited. Yet, this level of instruction was sufficient for computer-literate adults to figure out what to do without further aid. We found that the children, in general, did not bother to read help screens anyway or even the simple instructional messages that appear on the playground or game pages. The kids were far more likely to call out to someone else in the room and ask for assistance.

The children were able to choose between the games of Keyit and Pickey for the entire test period. The games Automath and Monkey were available for the

final few sessions. All these games are described in chapter 3.

The purpose of the pilot testing was to perform formative assessment of CEL, to ensure that the system was accessible from a real school setting, that the player client was usable by children and that the children enjoyed their experiences with the system. During the initial testing period, the user interface was adjusted as described in section 4.7.

The remainder of the period was spent validating CEL's data capture and storage mechanism. The goal was to demonstrate that CEL can collect the types of data common to the ILS field and that this data can support the types of analyses generally performed by researchers in this field.

Data was collected during pilot testing for 19 days¹. Figure 6.1 plots the amount of time that data was collected each day. Typically, we were given an hour with each class, which included time to set up the lab, organize the children and take them to the lab, followed by start-up time for them to log in and begin playing games.

¹Note that on days 3, 4 and 6, various system problems occurred and we had to curtail data collection. Twice, the school's Internet server went down. Once the Brandeis network went down.

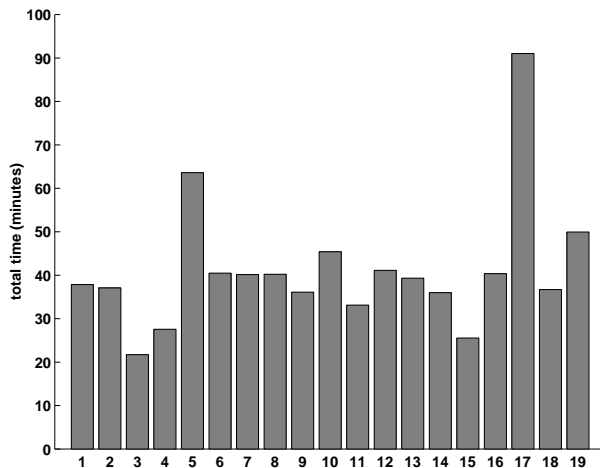


Figure 6.1: Data collection time, per day.

Next, we illustrate our claim that the data collected in CEL can support the types of analyses generally performed by researchers in the ILS field, as described in chapter 2. Analysis that supports four of these categories is shared in this section: *activity*, *interaction*, *learning* and *interest*. The fifth category of analysis, *coverage*, is the subject of chapter 8.

We have selected four students to serve as examples throughout, based on their representative amount of activity with CEL and performance statistics. Figure 6.2 illustrates the number of words completed versus average typing speed for each student in the group for the games of Keyit and Pickey. The sample students are highlighted. Note that we have chosen one fourth grade boy (id = 119), one fourth grade girl (id = 98), one fifth grade boy (id = 88) and one fifth grade girl (id = 89).

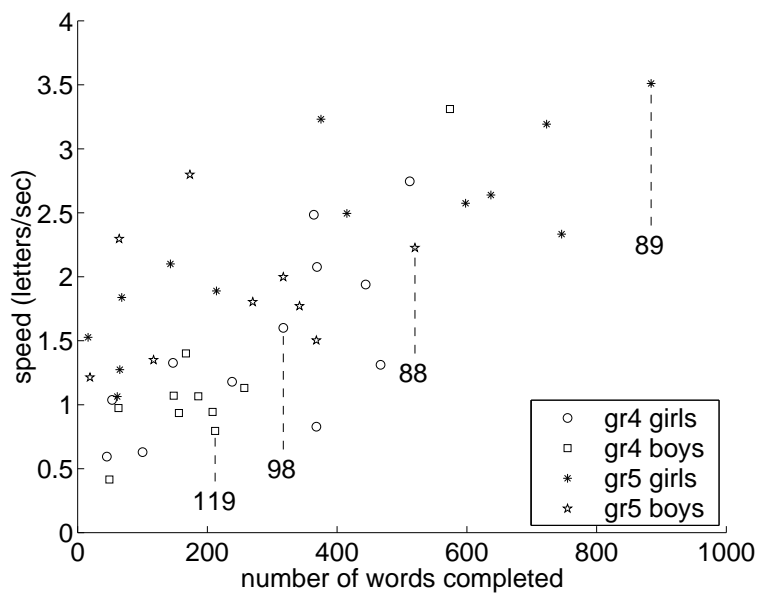


Figure 6.2: Number of words completed versus typing speed, per student.

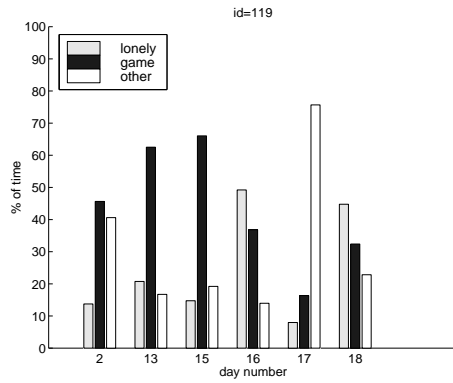
6.1 Activity

One common analysis concerns examination of participants' activities when using an interactive learning system. Researchers and teachers want to know what users are doing with their time on-line. In CEL, students engage in three main activities: (1) playing games ("game time"), (2) sitting in playgrounds waiting for something to do ("lonely time"), and (3) doing other things, like editing IDsigns ("other time"). This data is available in the user behavior tables (see chapter 5). An example is shown in table 6.2, which contains the amount of time spent in each of these activities for one of the children involved in the pilot testing (id = 89).

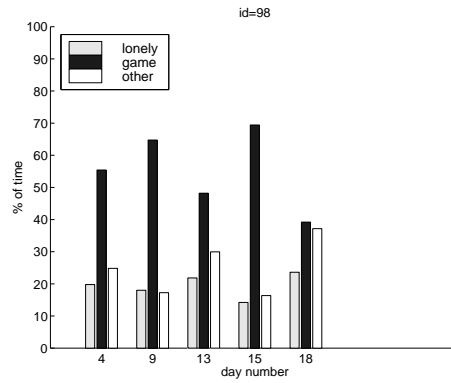
Table 6.2: Activity statistics.

day	start time	end time	elapsed time (sec)	lonely time (sec)	game time (sec)	other (sec)
3	11:12AM	11:32AM	1233	424 (34.39%)	134 (10.87%)	675 (54.74%)
5	10:58AM	11:31AM	1996	890 (44.59%)	552 (27.66%)	554 (27.76%)
8	11:13AM	11:36AM	1346	406 (30.16%)	649 (48.22%)	291 (21.62%)
10	11:47AM	12:01PM	884	294 (33.26%)	383 (43.33%)	207 (23.42%)
12	11:21AM	11:58AM	2164	464 (21.44%)	656 (30.31%)	1044 (48.24%)
17	10:35AM	11:55AM	4841	1418 (29.29%)	1731 (35.76%)	1692 (34.95%)
19	11:33AM	11:57AM	1434	269 (18.76%)	425 (29.64%)	740 (51.60%)

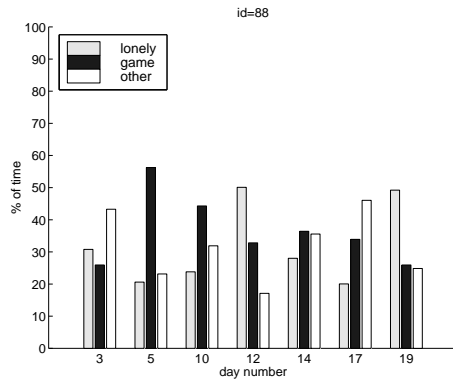
Figure 6.3 shows this type of data graphically, for the four students in our sample group. The horizontal axis contains the days that the students participated. One day's activity is represented by a group of three vertical bars, indicating the amount of time that this student spent in "lonely", "game" or another state.



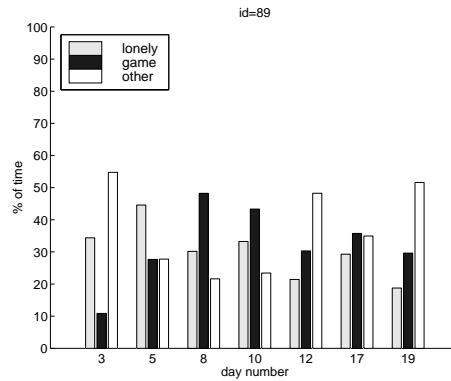
(a) id = 119



(b) id = 98



(c) id = 88



(d) id = 89

Figure 6.3: Activity charts for sample students.

A summary of the activities for all the students is illustrated in figure 6.4. One might be interested in looking at the amount of time spent playing games versus the amount of time sitting in the playground.

Examination of this type of data across all students may help researchers and system builders determine which elements of a system are more attractive to students than others. For example, on 5 of 19 days, more than half the overall time was spent playing games. On 13 of 19 days, more than a third of the time

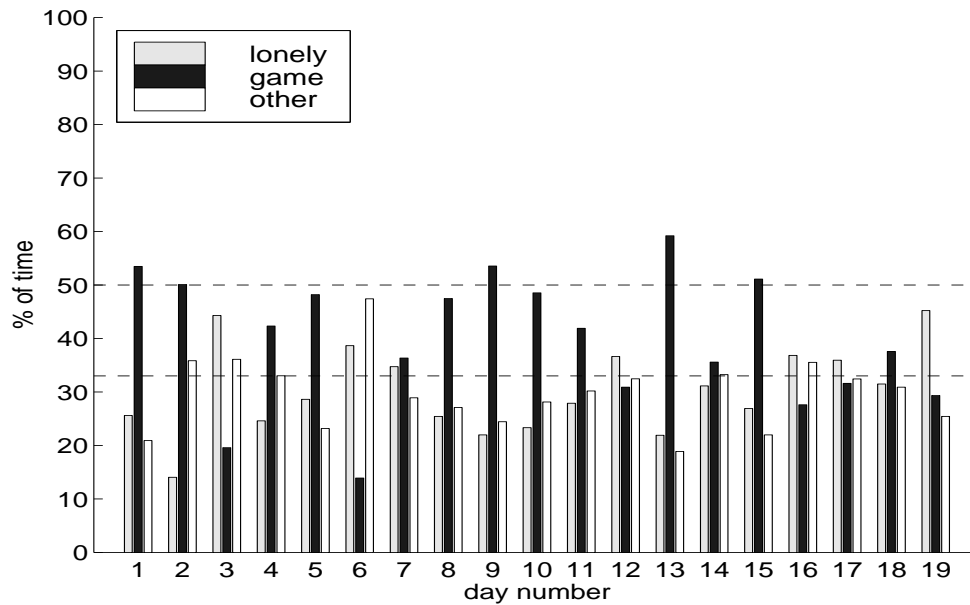


Figure 6.4: Summary activity chart.

was spent playing games.

Analysis of this data on an individual basis may help teachers find out more about their students' needs. For example, student 98 (see figure 6.3(b)) spent more time playing games than sitting in the playground or doing other things. However student 89 (figure 6.3(d)) spent more time doing other things — primarily editing her IDsign. Since this student was also the fastest typer, these types of analyses can indicate to a teacher that perhaps this student needs to be challenged more.

6.2 Interaction

In multi-player games, researchers and teachers are interested to find out who participants are interacting with. As well, if software agents are involved, system builders want to know if students are interacting with the agents and be able to compare human-human with human-agent encounters.

During pilot testing, the students interacted with both human playmates and software agents. To test the various modes, we varied the interactions on each day, as summarized below:

IN – Internal interaction.

All participants that were logged into CEL were present in the computer lab.

EX – External interaction.

Several participants were humans not present in the computer lab – some were other children logged in on classroom computers elsewhere in the same building, and others were graduate students logged in at various locations outside the primary school.

SA – Secret agent interaction.

Some “participants” were simple “secret” software agents that were coded to enter a playground, accept challenges and play games, letting their human opponents win at least 50% of the time.

Table 6.3 shows which participants were involved during each test day. On some days, more than one type of interaction occurred.

Table 6.3: Summary of interactions.

day	participant(s)	number of games played	number of kids	grade level(s)
1	EX	75	11	4,5
2	SA	70	7	4
3	SA	11	5	5
4	SA	37	7	4
5	EX+SA	149	14	5
6	EX+SA	12	6	4
7	IN	67	9	4
8	EX+SA	111	12	5
9	SA	128	10	4
10	SA	120	9	5
11	EX	60	8	4
12	EX+SA	88	9	5
13	EX	117	9	4
14	EX	85	7	5
15	IN	52	12	4
16	SA	54	9	4
17	EX	227	25	4,5
18	IN	61	12	4
19	EX	48	9	5

Analysis of interaction conditions varies, depending on the goals of individual experiments. Some of the types of studies that might be performed include the change in activity rate under different interaction conditions, the amount of interaction between different types of participants and the specifics of who interacts with whom. The next set of figures illustrate these analyses for the data collected during pilot testing. Note that these graphs do not take into account the number of participants available in each category, nor was a control study performed, so the reader is cautioned not to draw any specific conclusions concerning the success or failure of the different interaction conditions.

Figure 6.5 shows the rate of game play under the various conditions, i.e., the number of games played per minute, averaged across all the students participating that day.

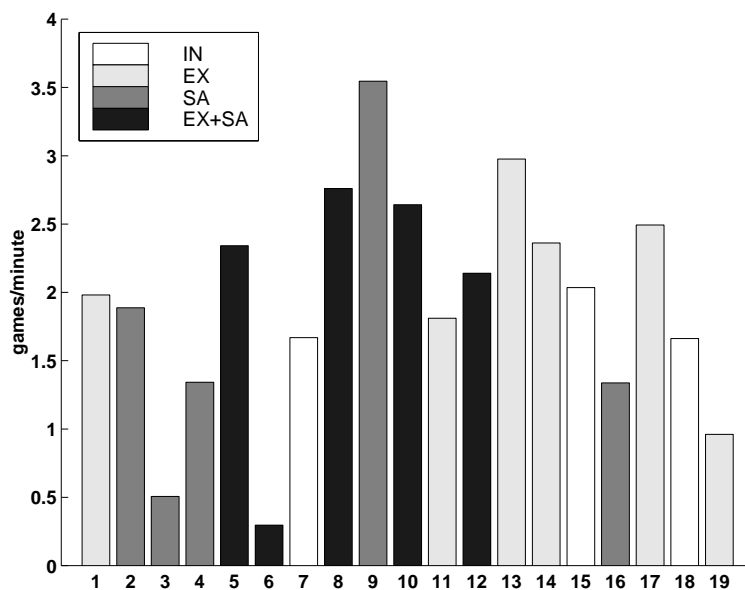


Figure 6.5: Number of games played per minute.

Figure 6.6 contains a breakdown of the games played by the kids amongst themselves (IN), with the secret agents (SA) and with others who were not in the room (EX).

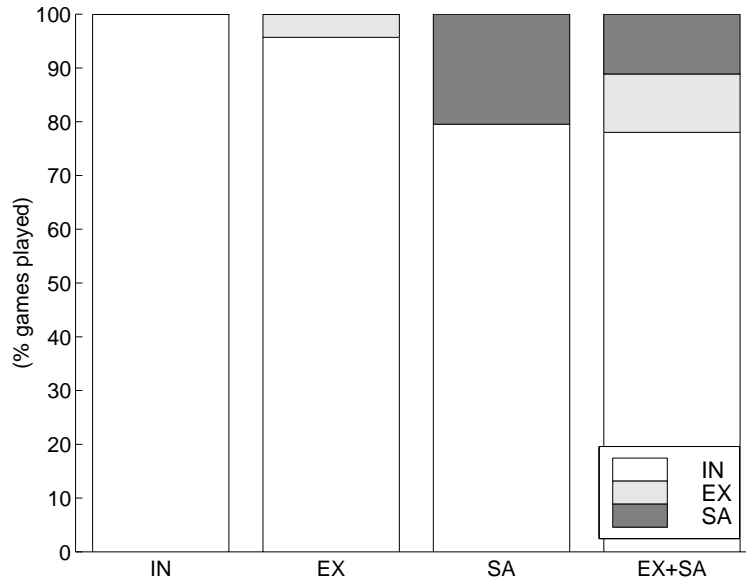


Figure 6.6: Interactions between types of participants.

Figures 6.7 and 6.8 illustrate the interactions between the various factions of players. In both graphs, the vertical axis shows the players who initiated matches and the horizontal axis lists the players who accepted. A point indicates that at least one encounter occurred between that given pair of players.

In figure 6.7, the children are grouped according to grade level and gender; the same order is used on both axes. It is clear to see that the two classrooms of children were infrequently in CEL at the same time (which was due to scheduling constraints, as stated earlier), thereby verifying our methodology.

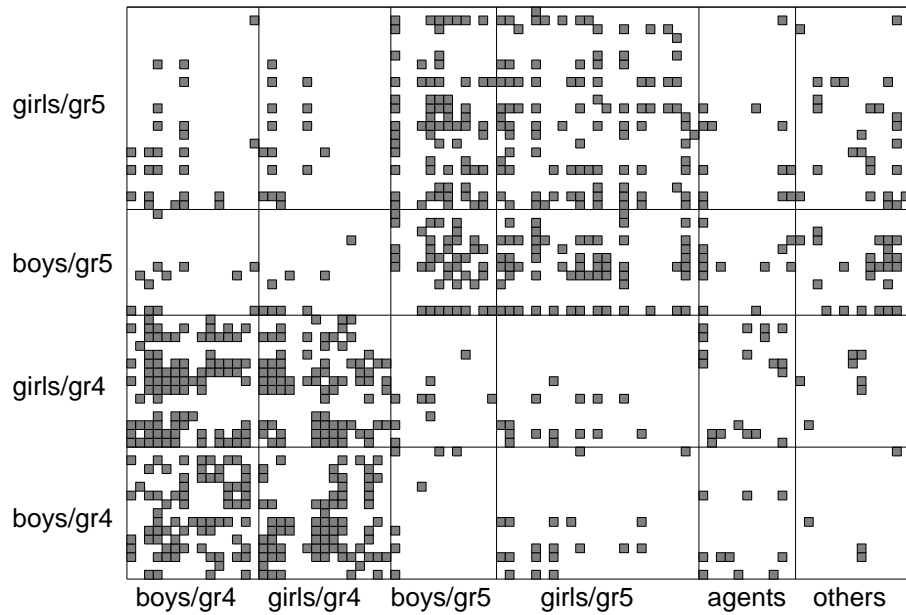


Figure 6.7: Who plays whom, grouped by age and gender.

Figure 6.8 contains the same data, but organized differently. The four “kids” sections of figure 6.7 are intermingled, and the children are ordered by typing speed as recorded during Keyit and Pickey games. The fastest typers are nearer to the origin.

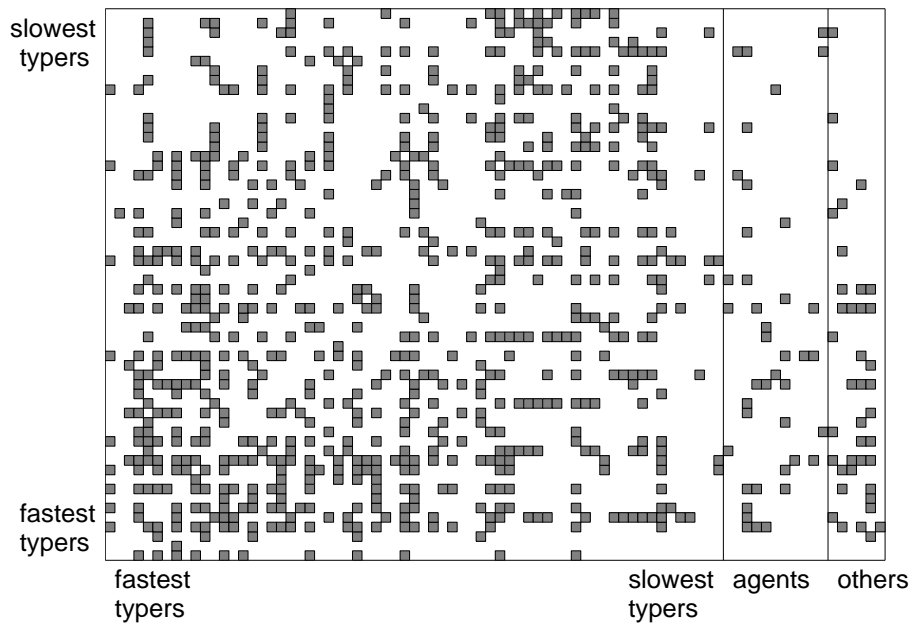


Figure 6.8: Who plays whom, ordered by typing speed.

This type of analysis can prove extremely useful in an open Internet system. Students of different ages, genders and locations may interface with each other, and plots like these can help researchers identify clusters of interaction. These plots may also help highlight the acceptance rates of software agents amongst different segments of the human population.

6.3 Learning

Naturally, everyone wants to know if participants' performance improves when using an interactive learning system. CEL can track changes in performance.

As an example, we look at the change in the children's typing speed as measured at the beginning and the end of pilot testing. Figure 6.9 plots the change in typing speed for two of the students in the sample group. The horizontal axis represents time, in terms of the number of words typed. The vertical axis represents typing speed, in letters per second. We have normalized both axes for all the students, in order to make comparisons easily. In the horizontal dimension, this allows us to take into account the number of games each student played in CEL.

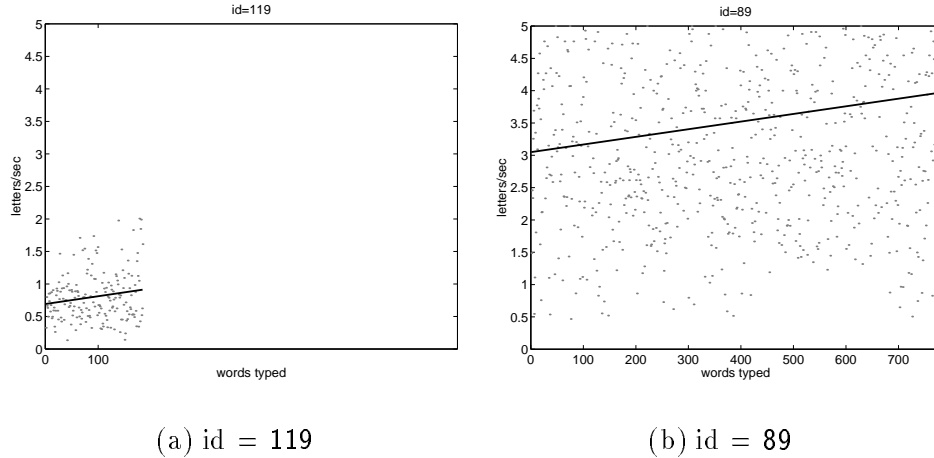


Figure 6.9: Tracking learning in sample students.

Figure 6.10 summarizes the change in typing speed for all the students. A plot like this, read in conjunction with pre- and post-test results, can give

researchers and teachers alike a good indication of the learning that has occurred while students use a particular system.

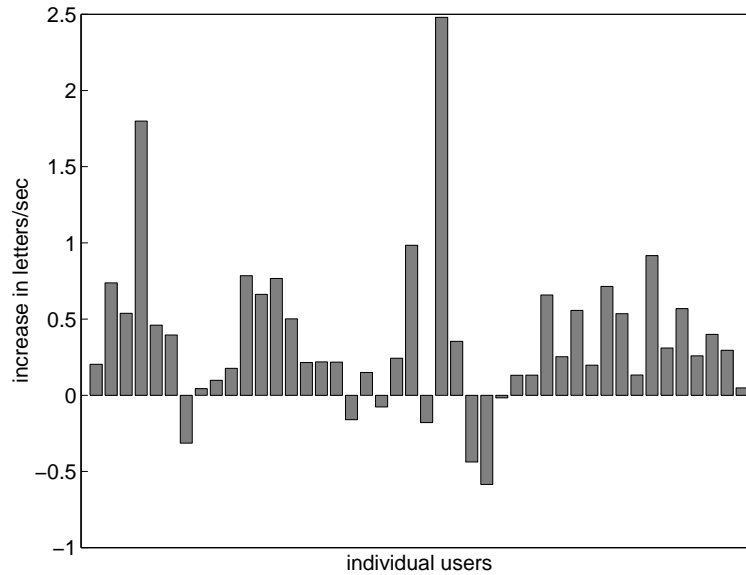


Figure 6.10: Change in typing speed.

Of course, this data taken alone is not very meaningful. Control studies need to be performed in order to prove that learning has been a direct result of using a system. For example, chapter 8 discusses use of an alternative engine for providing content to games of Keyit and Pickey. Future work in that chapter highlights the need for a controlled experiment, where students are divided into three groups and three different game content selection methods are compared, one per group. Having plots like figure 6.10 can help highlight any difference in students' learning rates between the three groups.

6.4 Interest

Determining participants' interest levels in an interactive learning system is quite important to researchers and system builders. In CEL, participants can indicate their interest by completing an on-line survey, which is presented upon leaving a game playground (see figure 6.11). Users are asked to rate their matches on two scales: *enjoyment* and *difficulty*.

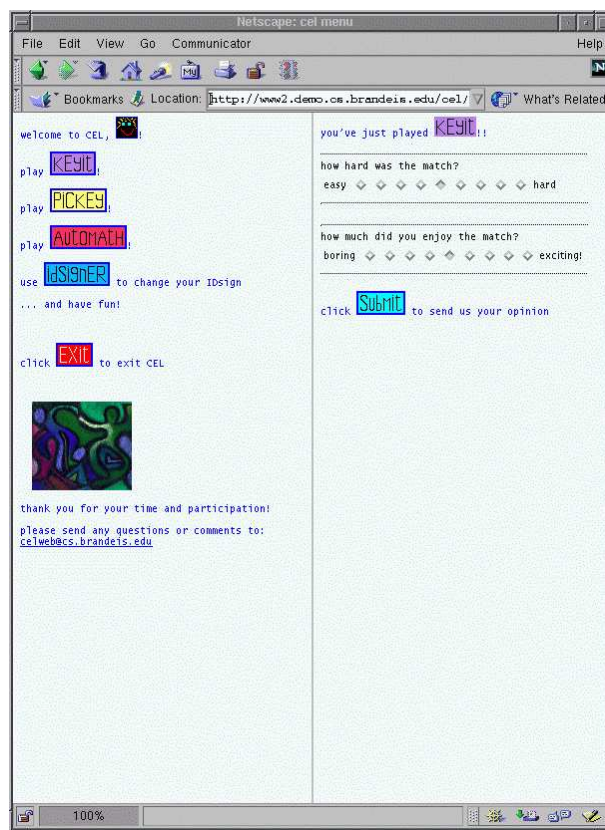


Figure 6.11: Exit poll.

As an example, we plotted the data collected during pilot testing in figures 6.12 and 6.13. Although answering these questions is optional, 90% (40 out of 44) of the students completed the on-line survey.

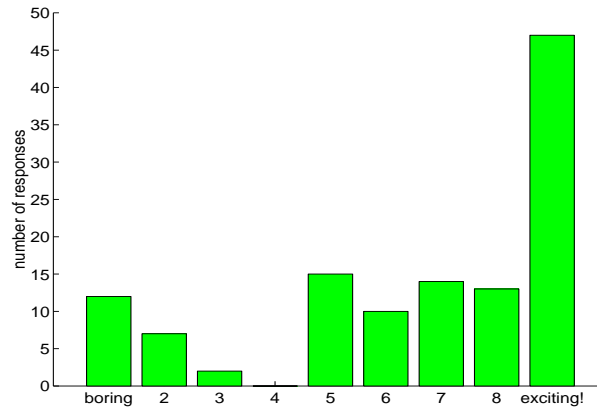


Figure 6.12: “How much did you enjoy the match?”

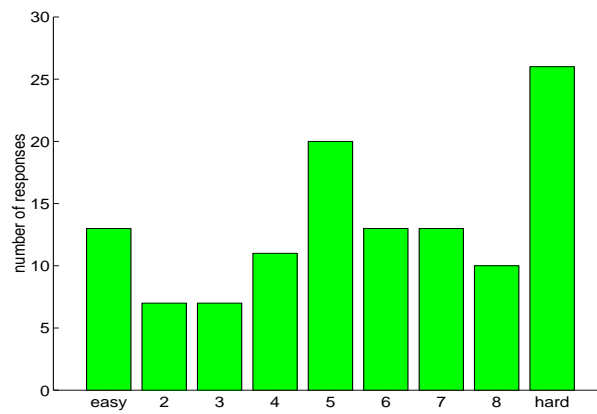


Figure 6.13: “How hard was the match?”

6.5 Off-line survey

At the end of the pilot study, we asked each of the children to complete a short, anonymous paper-and-pencil survey in order to get some feedback from the children about their experiences with CEL. The survey contained six questions, and the wording was geared toward fourth and fifth graders. Forty of the surveys were returned (90%), from 22 of the fourth grade children and from 18 of the fifth grade children. The survey is shown in figure 6.14.

CEL Survey	Spring 1999
You have been a participant in CEL (Community of Evolving Learners) this year. CEL is a research project that is supposed to help learners of all ages come together on the Internet. We would like you to tell us what you think of CEL. Your answers will help us improve the system for next year.	
Please answer the following questions honestly and do not write your name on this survey. Thank you!	
(1.) What grade are you in?	_____
(2.) Which games did you play in CEL?	_____
(3.) Name 3 things that you like about CEL.	_____
(4.) Name 3 things that you don't like about CEL.	_____
(5.) Name 3 things that could be changed to make CEL better.	_____
(6.) Is there anything else you would like to tell us?	_____

Figure 6.14: Post-study survey.

The survey results for questions 3 through 5 are tallied in tables 6.4 through 6.6, respectively. The tables should be read as follows: if 21 students said they liked creating IDsigns, this means that 21 of the children chose to include this feature in their list of things they liked (it does not mean that the remaining 19 children did not like creating IDsigns).

Table 6.4: Name 3 things that you like about CEL.

creating their IDsign	21 children
learning (to type faster)	21
contacting others	15
anonymity	8

Table 6.5: Name 3 things that you don't like about CEL.

nothing!	15 children
when it breaks	8
matches were too hard	7
not enough games	6
players can cancel matches	3
not exciting enough	2
anonymity	2
non-anonymity	2
matches were too easy	1
score reporting is confusing	1

The children suggested a few interesting improvements, such as allowing users to select from a set of ready-made IDsigns, in case they do not want to make their own. We found it ironic that, despite our sensitivity to open competition in an educational setting, several children's suggestions involved making the site feel more competitive: e.g., adding skill levels, ranking pages and prizes. Although

Table 6.6: Name 3 things that could be changed to make CEL better.

add more games	16 children
nothing	9
fix it so it would never break	6
make matches harder	5
make matches easier	5
make it run faster	3
make better games	3
add computer opponents	3
create skill levels	3
add a ranking page	3
prevent players from canceling matches	2
allow practice sessions	2
be able to pick a ready-made IDsign	1
get prizes	1
get free stuff on the site	1
make 3, 4, 6-player games	1
add more color choices in the IDsigner	1
add a chat capability	1

the consensus for question 4 was that the matches were too hard, for question 5, an equal number of children commented that the matches were too easy and that the matches were too hard.

For question 6 (“Is there anything else you would like to tell us?”), 29 of the children wrote either “no” or did not respond. A few suggested improvements, which we tallied along with the responses to question 5. Seven children wrote statements emphasizing how much they liked the system (“It was really fun.”).

Here are some highlights from the survey²:

- A fourth grade child wrote: “I would like to play these games next year.”
- A fifth grade student wrote: “It was WAY cool.”
- A fourth grader wrote that s/he would like it if s/he “could bye [*sic*] free stuff there.”
- A fourth grade child wrote that s/he liked the fact that s/he “could be who you want to be with the picture [*sic*]” (i.e., the IDsign).
- A fourth grader wrote: “It helped me to improve my consintration [*sic*].”
- Two students commented that it is easy to use and to understand.
- Several children wrote that they liked that they could play their friends (these comments were included in the “contacting others” figure in table 6.4).

In summary, we found that the children quite enjoyed using CEL. Indeed, the most common response to question 4 (“Name 3 things that you don’t like about CEL”) was the statement that there was nothing they didn’t like about CEL.

²Note that this school practices “Write to Read”. With this method, early writers are taught first to express themselves and second to focus on mechanics like spelling, punctuation and grammar.

6.6 Summary

Pilot testing is a useful method for system builders to evaluate the mechanics of their system and for researchers to confirm that the types of data they would like to collect are being gathered and can support the claims they would like to make with their research. Once pilot testing is completed successfully, experimental research can begin in earnest.

With CEL, the early phase of pilot testing was spent doing formative assessment of the system. Many adjustments to the user interface were made during this period (see section 4.7.1). Subsequently, data was collected, and the plots shown in this chapter demonstrate that the types of analyses generally performed by the ILS field (as laid out at the end of chapter 2) can be supported by this data.

Chapter 7

Agents as learning partners

CEL is a multi-user environment, and the activities inside the system are multi-player games. If not enough people are logged into a playground, then we let software agents act as artificial learning partners, maintaining an active presence in the system at all times and thereby sustaining the community. The software agents in CEL are designed with a flexible architecture so that different control mechanisms can easily be used to implement agents exhibiting a variety of behaviors. We refer to the agents as *secret agents* because there is no explicit means for participants to distinguish the artificial players from other humans.

This chapter begins with a functional description of CEL agents and outlines the types of behaviors that the agents need to embody. Two behavior modules are discussed, one for playground behavior and one for game behavior. To demonstrate the flexibility of the CEL agent architecture, we have built two different control mechanisms for game behavior. During pilot testing (described in chap-

ter 6), we used agents that exhibit simple rule-based behaviors. Subsequently, we took data collected during the pilot study to train more sophisticated agents controlled by neural networks. We describe the control modules for both types of agents and present training results.

7.1 Functional description

Inside CEL, secret agents may exhibit three categories of behaviors:

1. system behavior,
2. playground behavior, and
3. game behavior.

System behavior refers to high-level actions like logging into and out of CEL at particular times of day and selecting different playgrounds. Currently, we have not implemented a mechanism for this level of behavior, so we leave this for future work and here only discuss playground and game behaviors. Playground behavior refers to entering and exiting playgrounds and initiating challenges. Game behavior refers to the play within a specific game.

A top-level controller for the secret agent decides which behavior to follow, as indicated in figure 7.1. A different module handles each type of behavior. The module that operates at any given time is determined based on the current state of the agent, defined on the basis of the input received from the CEL server.

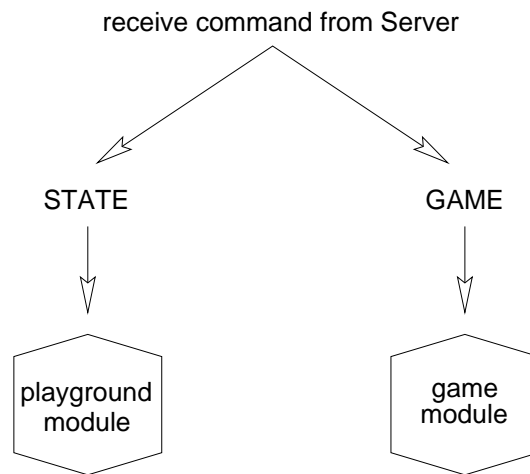


Figure 7.1: Basic control architecture.

Figure 7.1 is highly simplified because there are more than two commands that can be received (i.e., other than **STATE** and **GAME**). The complete set of commands is listed in table 7.1, along with the requisite responses from the agent. The precise handling of game commands varies, depending on which game is being played. The general and playground commands are handled the same way, no matter which game playground an agent is residing in.

Table 7.1: Commands sent to agent from CEL server.

	command received from server	response from agent
<i>general commands</i>	SHUTDOWN	exit
	LOGOUT	send LOGOUT command to server and exit
	PONG	no response
	PING	send PONG command to server
	SETTIMEOUT	change timeout value
	GETTIMEOUT	send timeout value to server, in the form of a GETTIMEOUT command
	ERROR	exit
<i>playground commands</i>	STATE	update internal state; wait; and then send STATE or ASK command to server
	REJECT	update internal state; wait; and then send STATE or ASK command to server
<i>game commands</i>	GAME	this means agent has been invited to play a match, or agent's invitation to another player has been accepted — now agent must wait for game data to arrive from server, in the form of an DATA command; $state \leftarrow GAME_STATE$
	DATA	this command contains game data; response is game dependent
	MOVE	this command contains opponent's move; response is game dependent
	BADMOVE	this means agent's move was invalid; response is game dependent
	RESULTS	this means opponent has finished match; response is game dependent
	ABORT	abort match; wait; then send STATE command to server; $state \leftarrow PLAYGROUND_STATE$

7.2 Playground behavior

There are three basic choices for things to do in a playground:

1. invite a playmate to engage in a match
2. wait to be invited by another player
3. leave the playground

We note that human players may also perform unexpected actions, such as clicking on the “back” button of their browser, selecting a previous page from their browser’s history list or closing their browser without formally exiting from CEL. We do not model these types of spurious behaviors in the software agents and only stick to so-called “normal” expected actions.

The client software used by human players updates its playground state every 5 seconds if the human does not initiate an action herself. (The mechanism for accomplishing this is detailed in chapter 4.) Thus, if the human player sits at her computer while in playground mode and does not click on another player to initiate a match, then every 5 seconds her browser sends a **STATE** command to the CEL server and in return, she receives an update message. This message is either a **STATE** command, indicating the present state of her playground (i.e., a list of the playmates who are currently logged into her playground), or a **GAME** command, if another player has invited her to engage in a match¹. If the human

¹If a system error occurs, then an **ERROR** command may be received, or if the system is shutting down, a **SHUTDOWN** command may be received.

player clicks on another player to initiate a match, then an **ASK** command is sent from her browser to the CEL server. Figure 7.2 illustrates the client’s actions.

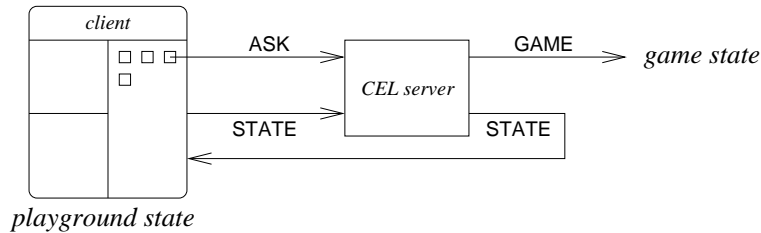


Figure 7.2: Playground behavior.

The software agent follows a similar operating procedure. The control architecture decides between the three options (listed at the beginning of this section), and if the choice is to wait to be invited by another player, then the agent waits 5 seconds and sends a **STATE** command to the CEL server. Otherwise, the agent sends an **ASK** command to the server, specifying a playmate to invite for a match.

Creating an agent that acts in a playground is relatively straightforward. All we really need to model is the likelihood of a player inviting a playmate to engage in a match (i.e., when a client sends an **ASK** command to the CEL server). We could use human data to determine these probabilities. For example, we tallied the probability of players generating **ASK**, **STATE** and **EXIT** commands for each of the forty-four participants in the pilot study, as shown in figure 7.3. We could use these probabilities as the basis for controlling forty-four different playground agents.

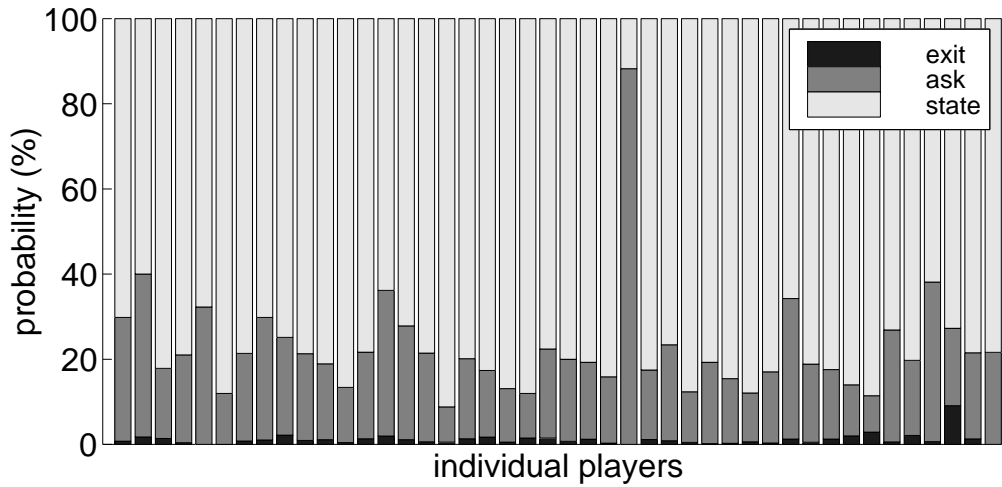


Figure 7.3: Command probability.

7.3 Game behavior

Game behavior is of course dependent on the particular game being played. Additionally, different humans will exhibit varying levels of ability as well as various characteristics of play.

Figure 7.4 illustrates the command sequences exhibited while in a game state, by either human or secret agent players. While playing a game, a sequence of MOVE commands flows between the player, the CEL server and the player's opponent or partner (depending if the game is competitive or collaborative).

Note that this drawing (figure 7.4) is somewhat simplified, as the game's matchmaker is also involved. Each MOVE command received by the CEL server is passed along to the matchmaker, where the move is evaluated. If the move is legal,

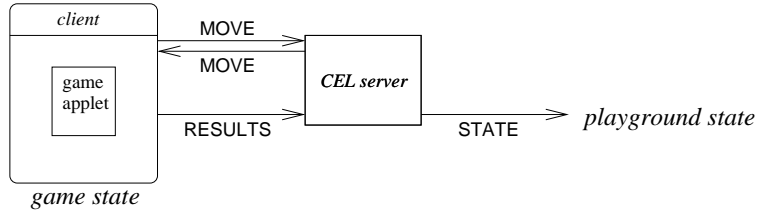


Figure 7.4: Game behavior.

then the matchmaker sends the same **MOVE** command back to the originating player and also to the player's mate. If the move is illegal, then the matchmaker generates a **BADMOVE** command and sends that only to the originating player. When the game is over, the **RESULTS** command is also forwarded by the CEL server to the matchmaker, to mark the completion of the match.

The content and timing of **MOVE** commands is specific to each game, according to the rules of play. We use Keyit as an example. The game play for Keyit is as follows: the system provides ten words to each player, one at a time, to type as fast as she can, with 100% accuracy. Players participate asynchronously, completing each word at their own pace.

During pilot testing, we implemented a very simple Keyit agent that waits until the human has typed each word; then the agent pauses for a variable amount of time and returns its score² for the same word. The agent's score is calculated based on the human's performance and is chosen to be slightly faster or slower than the human's. This is done to create the motivational illusion that the

²Score is the time it takes to type the word, in hundredths of a second.

human's opponent is of comparable ability to the human and so that the human will win matches with these agents approximately half the time.

While these simple Keyit agents performed adequately during the pilot study, it is obvious that their restricted capabilities limit their usefulness as effective learning partners. Because the architecture for CEL agents is flexible, we can easily substitute a more sophisticated control mechanism. The remainder of this chapter discusses just such a control mechanism which is designed to emulate the behavior of humans.

7.4 Training agents to emulate humans

An earlier project provided us with background for extending the simple Keyit agents to exhibit more sophisticated, human-like behaviors. In this section, we first describe the earlier project and then detail our methodology and results from extending that work to Keyit.

In a follow-on project to the Tron Internet experiment (mentioned in section 3.3.6), we trained agents to play Tron, with the goal of approximating the behavior of the human population in the population of trained agents [Sklar et al., 1999]. These agents were controlled by feed-forward neural networks and were trained using *supervised learning* [Pomerleau, 1993; Wyeth, 1998].

We trained the Tron-playing agents using two methods. First, we trained agents to emulate the performance of individual humans, based on a one-to-one correspondence between human trainers and network trainees. Second, we

trained agents to emulate the performance of a small population of humans, grouped together according to a similar performance statistic. This scenario was based on a many-to-one correspondence between human trainers and network trainees. The training procedure involved replaying Tron games, allowing the trainees to observe and predict moves. Based on the accuracy of the trainees' predictions, the network weights were adjusted using the backpropagation algorithm [Rumelhart et al., 1986].

We evaluated our training efforts in two ways. First, we compared the performance of the networks to their trainers', looking for correlations. We found that it was very difficult to train a network to emulate the precise behavior of an individual human. However, the performance of the population of trainees was comparable to the performance of the human population. We speculated that some of the artifacts of the Tron domain likely contributed to the discrepancies between the artificial trainees and their human counterparts.

Second, we examined the effectiveness of our method as a general means toward training software agents. It is important for artificially trained agents to experience a wide variety of behaviors, otherwise they will not be robust and will only perform well in situations similar to those experienced during training. The conclusion drawn is that a population of humans can act as effective trainers for a graded population of agents, because there is naturally a wide variation in behaviors both across an entire population of humans and within a single stochastic human player.

As well, the training procedure proved to be a valid technique for capturing

regularities within a large database of game moves. Indeed, we found that some of the trainees performed better than their trainers. In the game of Tron, if a match lasts for n moves, a player can make $n-1$ good moves and then lose the game with one bad move. When such games are observed by trainees, they learn more “good” play from the $n-1$ moves than they do “bad” play from the one false move. In this way, the training method serves to filter out infrequent mistakes of the human trainers.

Now we carry these techniques into the CEL domain. We used the data collected during the pilot study as the basis for training agents to play Keyit. The control architecture, training methods and results are detailed in the following pages.

7.4.1 Architecture

In Keyit, the basic task can be described as follows: given a word, characterized by its corresponding set of seven feature values (described in chapter 5), output the length of time to type the word. In addition to the 7 feature values, we also consider the amount of time that has elapsed since the previous word was typed.

The agents are controlled by feed-forward neural networks. The network architecture is shown in figure 7.5. There are 8 input nodes, corresponding to each of the seven feature values (normalized) plus the elapsed time (mentioned above). The elapsed time is partially normalized to a value between 0 and (close to) 1. There are 3 hidden nodes and one output node, which indicates the time to type the input word, in hundredths of a second.

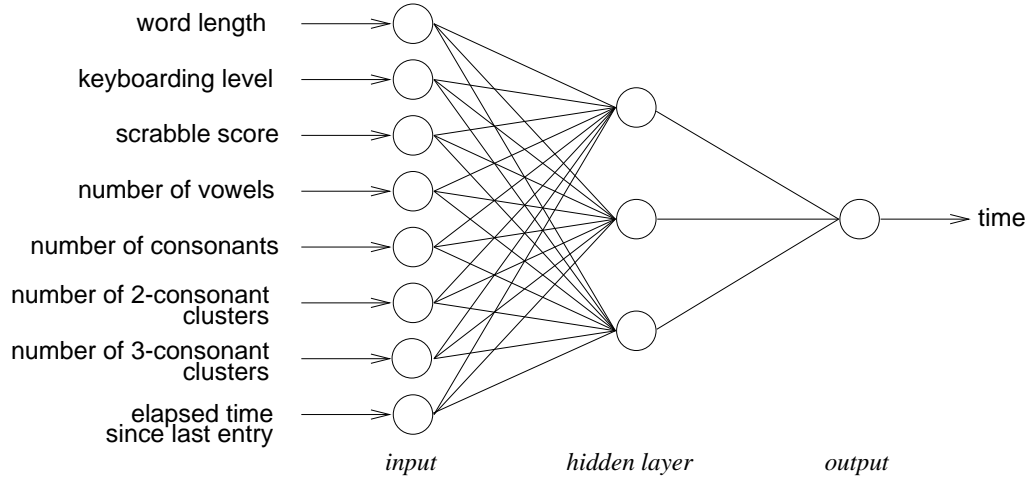


Figure 7.5: Neural network architecture.

7.4.2 Training

As with the Tron experiment, we trained players in two ways, first using a one-to-one correspondence between human trainer and network trainee, and second employing a many-to-one correspondence between groups of human trainers and network trainees. For the second method, we grouped human trainers according to their overall average typing speed.

For each human involved in the pilot study, we scanned the CEL log files, picking out all games of Keyit³, and gathered the moves from each game into a file, one per person. A “move” includes the timestamp (the time in seconds that the move occurred), the word being typed and the amount of time (in hundredths of a second) that the player took to type the word. Then we calculated the time

³Data from Pickey games was also used, because the two games are so similar.

elapsed between moves (based on consecutive timestamps) and, along with the seven feature values for each word, created two files (one for training and one for testing), placing alternate moves in each file.

We used all the data collected during the pilot study. The humans were learning throughout this period, so the networks were trained to approximate the average performance of each human across the entire time period. Figure 7.6 contains data for all the players involved in the pilot study, plotted in ascending order according to typing speed (in letters/second). The plot also indicates the groupings of players, used for the many-to-one training scenarios. The players are clustered according to typing speed, in increments of 0.5 letters/sec. The four players from our standard sample are highlighted.

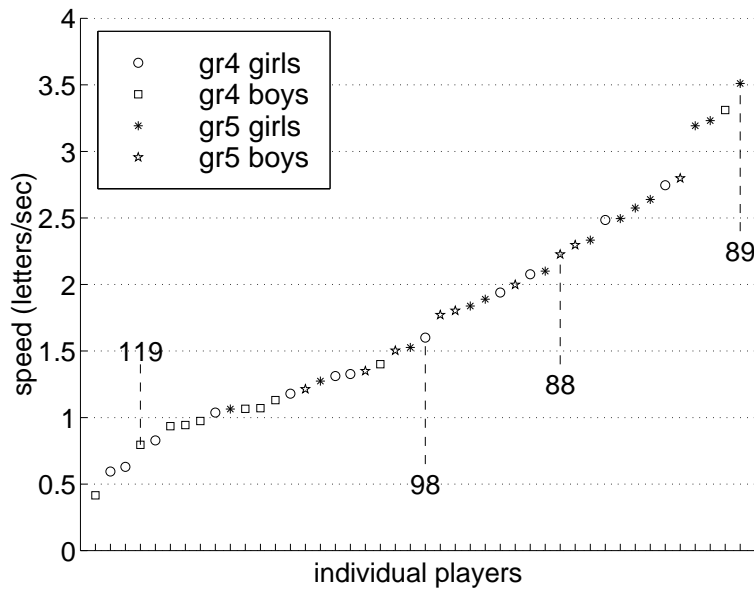


Figure 7.6: Average typing speeds of players.

We trained the networks using supervised learning, as in the Tron follow-on experiment, adjusting the networks during training using backpropagation. The results presented here were obtained with a learning rate of 0.00001. All the networks were trained for 10,000 epochs, but progress generally leveled off after 2500 epochs.

Throughout the training sequence, we kept track of the prediction error for the network — the difference between the typing time predicted by the network and the actual typing time of the training set. We saved one “best” network for each training sequence, corresponding to the set of weights which resulted in the smallest prediction error. After the training sequences were completed, we evaluated the best networks for each effort by comparing its prediction with the human’s data, for both the training set and the (reserved) test set of data.

7.4.3 Results

We look at the results of the training efforts in several ways. First, we look at the training period and show how the network improved its predictive ability during training. Figure 7.7 shows the performance of the networks trained for the four sample students (88, 89, 98 and 119). The plots in the top row illustrate the prediction error for the networks. The solid curve plots the error based on the test data set; the dashed curve plots the error based on the training data set. The plots in the bottom row show how the error in typing speed improves over time, when the networks are confronted with the test data set (solid curve) and the training data set (dashed curve).

The networks learn quite quickly, sometimes within 500 epochs. It is interesting to note that in some cases, as with students 98 and 119, the difference in prediction error between the training and test data sets is relatively marked; however the difference in typing speeds is negligible.

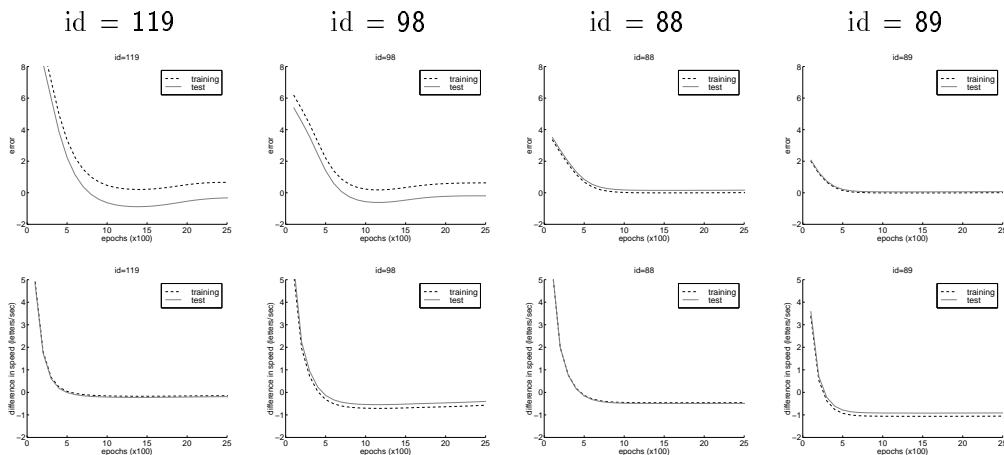


Figure 7.7: Improvement during training.

Another way in which we examine the training effort is by studying the correlation between the trainers and the best trainees. Figure 7.8 plots the typing speed for the trainees (horizontal axes) versus their trainers (vertical axes), for both the test and training data sets, for the one-to-one and many-to-one training efforts.

The correlation coefficients are listed in table 7.2, illustrating the average relationship between trainers and trainees across both populations. The correlation is much higher for the many-to-one trainees than the one-to-one trainees.

The final way in which we study the results takes a population-based ap-

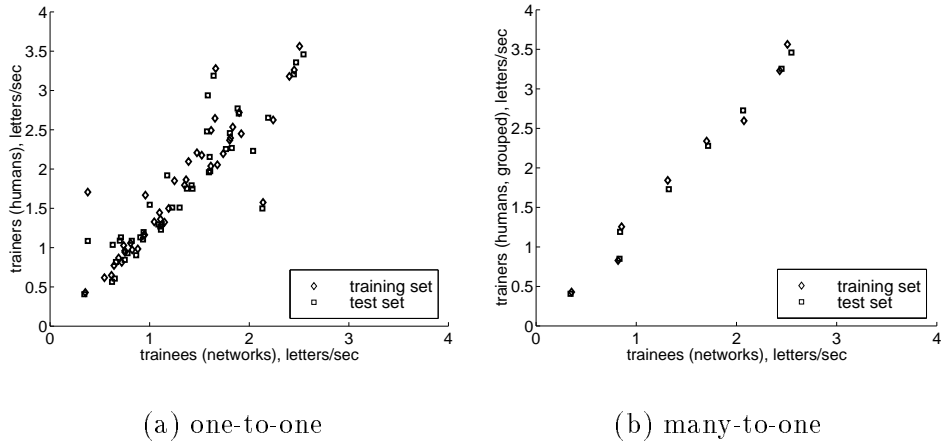


Figure 7.8: Correlation between trainers and best trainees.

Table 7.2: Correlation coefficients.

	training set	test set	population-based (test set)
one-to-one	0.636388	0.420413	0.800293
many-to-one	0.990960	0.996515	0.996515

proach. One objective with this project is to generate a population of agents, demonstrating a range of abilities. Figure 7.9 compares the average speeds of the human population with those of the agent populations, for both training schemes. The average speeds for the agent population were based on data collected during the testing runs only. Table 7.2 shows the correlation coefficients. In the one-to-one case, this population-based correlation is higher than the individual correlation; in the many-to-one case, the comparison is equivalent.

The comparison is made by first sorting both populations according to speed and then calculating the correlation coefficients. In the one-to-one case, sorting the trainees re-orders the comparisons that are made when computing the correlation coefficient, and so the correlation is higher. In the many-to-one case, the population-based correlation between trainers and trainees is precisely the same as in the individual case, because the training went so well that sorting the trainees does not change their order and so the two comparisons are equivalent.

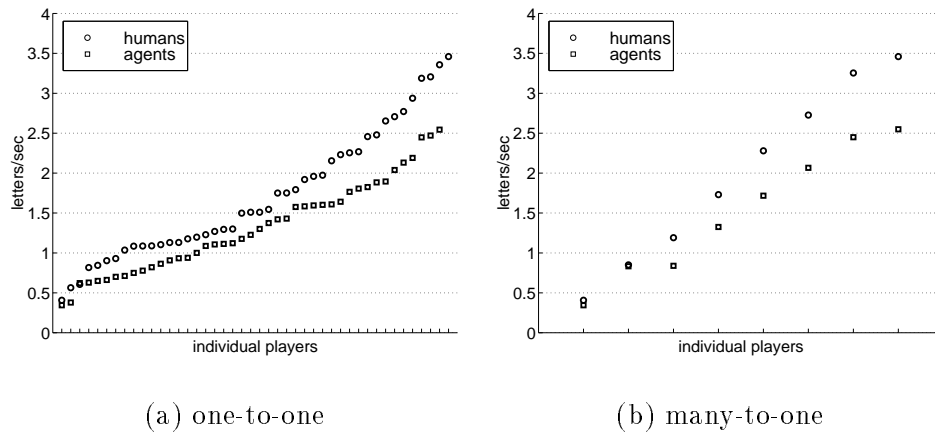


Figure 7.9: Correlation between populations of trainers and best trainees.

7.5 Discussion

In extending the simple Keyit agents to use a neural network controller, we were pleased to find that our training efforts in the CEL domain corroborated the earlier results of the Tron follow-on experiment. We summarize our observations. First, it is difficult to emulate exactly the behavior of individual humans. Second, it is better to approximate the behavior of a population of humans. Finally, it is best to train on a group of humans who exhibit similar features.

Future work will involve building agents that adapt their performance on-line. One method for accomplishing this would be to train an agent using data from the first few games, deploy the agent and then continue to train it further, by incorporating moves from subsequent games of its human trainer.

Future work will also involve adding complexity to the playground behavior module, in two ways. First, rather than using a rule-based controller where the frequency of issuing `ASK` commands is based on a fixed probability, a network controller will be implemented. This controller will be trained on the human data collected in CEL and will look at sequences of actions committed by the players being modeled. Second, a longer term project concerns making intelligent choices for which playmates to invite to engage in matches by incorporating student models to guide selection rather than choosing playmates randomly, as is implemented currently.

Chapter 8

Domain coverage

Both computer games and educational software provide an interactive medium with which humans can explore a domain. In the case of computer games, the domain might be outer space or the wild west or a fantasy land; with educational software, the domain might be arithmetic or geography or spelling. In either case, the purpose of an instructive interactive learning system is to guide a user through the domain in a methodical way, exposing him to as much of the domain as possible, without losing the user's interest.

This is typically done by characterizing each user according to his experience with the system and then moving him around the domain by providing a series of pre-defined challenges of increasing difficulty and/or complexity. With computer games, this methodology is often explicit; players must complete one "level" before being allowed to go on to the next. Although this format has proven to be highly motivating for both children and adults alike, the method does

not always provide an atmosphere dedicated to learning. Additionally, there are concerns about the appropriateness of such a formula for use in educational settings [Soloway, 1991].

The current trend in educational software moves away from pre-programmed and/or pre-leveled instructive environments, like traditional frame-based tutoring systems or leveled games, and towards constructivist environments where students are able to explore ideas for themselves without having to stick to fixed curricula [Papert, 1993]. In a classroom setting, this notion has been described as *learner-centered learning*:

A teacher is no longer a dispenser of knowledge addressed to students as passive receptors. Instead, where small teams of students explore and work together and help one another, a “teacher” becomes a colleague and participating learner. Teachers set directions and introduce opportunities. Teachers act as guides. [Forrester, 1992], p.11.

The same ideas can apply to educational software, where a software system acts as a teacher. The system should be adaptive and participate in the learning, guiding students through educational domains and adjusting as they advance.

The work described in this chapter brings adaptive behavior to educational games as a mechanism for supporting a learner-centered on-line environment [Sklar & Pollack, 2000a]. An evolutionary approach, guided by user performance, is used to select the content of matches for two simple keyboarding games, Keyit and Pickey (described in chapter 3). The purpose is to demonstrate the flexibility of CEL in being able to support experimental methodologies such as the adaptive approach used here.

8.1 The domain

In both Keyit and Pickey, players are presented with ten words to type. These words are selected from a database containing approximately 35,000 words. Every word in the database is characterized by a vector of seven feature values (these are detailed in chapter 5, section 5.1.1). Each word can be thought of as a point in this 7-dimensional feature space. Words with similar feature values are considered to be close to each other in this space; words with disparate feature values are considered far away. Figure 8.1 illustrates this for the word BLUE, which has close neighbors MEAT and BOIL. The words HIDE, DARK and RED are further away, respectively.

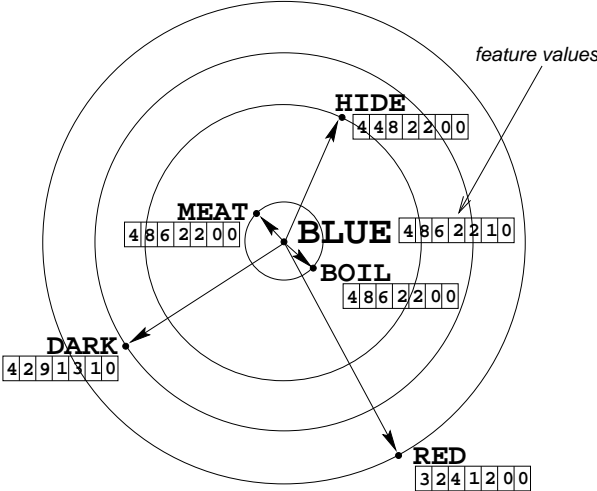


Figure 8.1: Distance between words in feature space.

8.2 Selection algorithm

An evolutionary approach is used to guide selection of words from the 7-dimensional feature space, geared to the changing needs of each individual user. The basic steps of a general evolutionary algorithm are outlined in table 8.1 [Holland, 1975]. The *elements* could be, for example, software agents exhibiting specific game strategies or various solutions to a hard search problem.

Table 8.1: Basic evolutionary algorithm.

- | |
|---|
| <ol style="list-style-type: none">1. Initialize a <i>population</i> of randomly chosen <i>elements</i>.2. Let each element perform in the task domain.3. Evaluate each element's performance and, based on the evaluation, <i>select</i> some elements to be replaced.4. Produce a new population of elements, using <i>reproduction</i> techniques like mutation and/or crossover to replace the elements selected in step 3.5. Iterate, starting from step 2. |
|---|

Our approach modifies this algorithm for the task of selecting words for Keyit and Pickey matches. In our context, the *elements* (referred to in table 8.1) are words, and the population size is fixed at 10. The modified algorithm is shown in table 8.2. Note that the algorithm shown here is in a simplified form, considering the needs of only one player; as described later in this chapter, the algorithm is further modified to accommodate the needs of two players. The algorithm is invoked by the matchmaker (see section 4.5 in chapter 4), when a player clicks

on the “start” button in a game applet. The selection and reproduction phases are illustrated in figure 8.2.

Table 8.2: Evolutionary word selection algorithm.

1. For a new user, initialize a population, G_0 , of 10 randomly chosen words.
2. For an old user, read the user’s performance data, which includes scores for all words previously encountered, the population of 10 words from the user’s last game (G_t), and an average score for all words the user has seen.
3. Evaluate the user’s performance with the words in G_t , and, based on the evaluation, select entries that are “known” and entries that “need practice.”
4. Produce a new population of words, replacing all entries in G_t to get G_{t+1} , using large mutations to replace “known” entries, and small mutations to replace entries that “need practice”.
5. Supply G_{t+1} to the user’s applet for the current game.
6. Iterate, starting from step 2, when the next game occurs.

The selection process (step 3 in table 8.2) involves comparing the score achieved for each word in G_t with the user’s average score over all words encountered in games of Keyit and Pickey. The idea is to partition G_t into two groups: those that the user knows how to type, and those that the user needs more practice with. “Score” is typing speed, calculated in hundredths of a second. Words whose score is lower than the average are deemed “known” (faster is better); words whose score is higher than average are labeled “needs practice”.

The reproduction phase (step 4 in table 8.2) entails replacing all the words

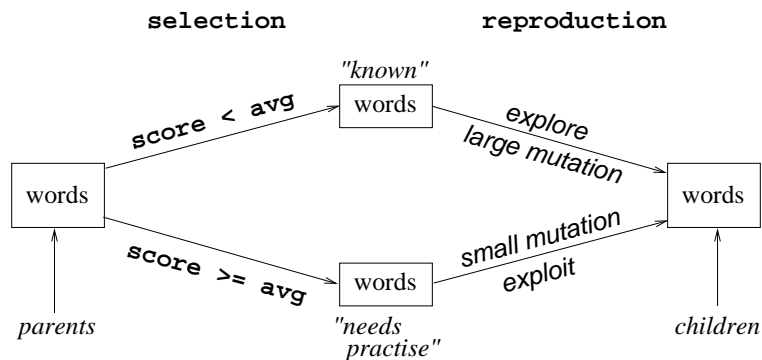


Figure 8.2: Selection and reproduction.

in G_t with appropriate *children*, to get G_{t+1} . “Needs practice” words are replaced with others nearby in the 7-dimensional space, thereby exploiting regions with similar feature values to provide more opportunities to master the similar words while avoiding repetition, where the same words might be offered again and again until they have been learned satisfactorily. This is equivalent to making a small mutation to a word’s feature vector. “Known” words are replaced by randomly jumping to some new area in the feature space, thereby exploring regions further away. This is equivalent to making a large mutation to a word’s feature vector. The general idea is illustrated in figure 8.3.

The actual implementation of this algorithm is complicated by two factors. First, when a game occurs between two human players, the set of ten words selected by the system must be appropriate for both players. Second, not all points in the 7-dimensional feature space are valid. If the reproduction phase used a standard operator like *mutation* or *crossover* and modified one or more values in

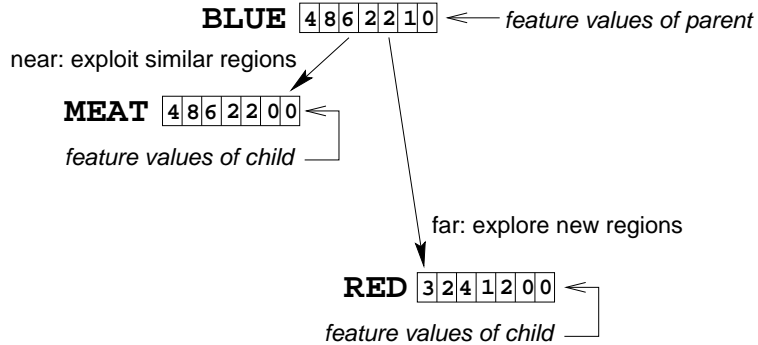


Figure 8.3: Exploitation and exploration in feature space.

a parent's feature vector, the resulting vector would not necessarily correspond to a word in the dictionary. In fact, some combinations of feature values are invalid, e.g., word length must equal the number of vowels plus the number of consonants. To address these complications, two procedures are introduced: merging and reproduction through sampling.

8.2.1 Merging

The merging procedure is implemented so that the contents of G_{t+1} is appropriate for both players engaged in the match. The basic process involves combining the user performance data for both players and creating a third, composite player that is essentially an average of the two players' performance statistics. In table 8.2, steps 1 and 2 are modified to read performance data for both players and then merge the data, so that steps 3 and 4 will be performed using this composite data set. Finally, step 5 sends the new population of words to both users' applets.

The data entities used by the merge process are detailed in chapter 5, section

5.2.2, and the notation used here is the same, with the following additions:

- P_1 = performance data for player 1
- P_2 = performance data for player 2
- P_c = performance data for composite player
- R = abbreviated notation for: $\langle n_r \rangle \langle r_0 \rangle \langle r_1 \rangle \dots \langle r_{n_r-1} \rangle$
- G = abbreviated notation for: $\langle n_g \rangle \langle g_0 \rangle \langle g_1 \rangle \dots \langle g_{n_g-1} \rangle$
- G''' = composite game data that is sent to both players' applets

G''' is analogous to G_{t+1} in step 5 of table 8.2. At the end of the merge process, it is guaranteed that G''' contains at least 10 and at most 20 words. At the end of the reproduction process, G''' will contain exactly 10 words. The algorithm is shown in table 8.3.

Table 8.3: Merging algorithm.

<ol style="list-style-type: none"> 1. Read user performance data for player 1: $P_1 \leftarrow \{R', G', \text{avg}'\}$ 2. Read user performance data for player 2: $P_2 \leftarrow \{R'', G'', \text{avg}''\}$ 3. Merge user performance data for player 2 with rates for player 1, creating one composite player: $P_c \leftarrow \text{merge}(P_1, P_2) = \{R''', G''', \text{avg}'''\}$ The source code for the <code>merge</code> function is shown in table 8.4. 4. If there are less than 10 elements in G''', then fill G''' with randomly chosen words until $G''' = 10$.

Table 8.4: Pseudo code for `merge()`.

```

function merge () {

  /* merge rates */
  R''' ← R';
  for i ← 1 to n_r''
    if r_i''.elemid is found in {r_0'''...r_{n_r''}'''} then
      (where r_j''' .elemid = r_i'' .elemid)
      r_j''' .count ← r_j''' .count + r_i'' .count;
      r_j''' .sum ← r_j''' .sum + r_i'' .sum;
    else
      n_r''' ← n_r'' + 1;
      r_{n_r'''-1}''' ← r_i'';

  /* merge game data */
  G''' ← G';
  for i ← 1 to n_g''
    if g_i''.elemid is found in {g_0'''...g_{n_g''}'''} then
      (where g_j''' .elemid = g_i'' .elemid)
      g_j''' .count ← g_j''' .count + g_i'' .count;
      g_j''' .sum ← g_j''' .sum + g_i'' .sum;
    else
      n_g''' ← n_g'' + 1;
      g_{n_g'''-1}''' ← g_i'';

  /* merge averages */
  avg''' ← (avg' + avg'')/2;

} /* end of function merge() */

```

8.2.2 Reproduction through sampling

The reproduction procedure must be able to take a parent and produce a child whose feature values are either near to or far from those of its parent, corresponding to *exploitation* (small mutation) and *exploration* (large mutation), respectively (as illustrated in figure 8.3). In theory, a traditional reproduction method like *mutation*¹ could be used for both tasks. To find a nearby entry in feature space, one of the parent's feature values could be selected at random and then incremented or decremented, to result in a new vector with only one value different from the parent vector. To find an entry far away in feature space, more of the parent's feature values could be altered, resulting in a new vector with values disparate from its parent.

As mentioned earlier, the problem with using this procedure in this domain is that the new vector would not necessarily be valid or correspond to a word in the dictionary. Some applications of evolutionary algorithms handle this kind of situation by applying a correction to the reproduction operator, ensuring that the result is valid. For example, a mathematical function (i.e., *modulo*) might be used to force the mutation of an individual feature value to fall within a specified numeric range. The situation here is complicated by the fact that even if individual feature values are valid, when taken in combination, the entire vector

¹For simplicity, the discussion here is limited to mutation. Crossover or other gene altering methods could also be used, but the problems encountered with using mutation in the present domain and real-time environment (as detailed in this section) would also occur with these other methods.

may be invalid. A simple method for overcoming this problem would be to try a series of mutations iteratively, stopping when a valid vector was found.

However, with this particular domain, the 7-dimensional feature space is quite sparse. If bounds are considered on each feature value (for example, word length must be between 2 and 25 characters, and keyboarding level must be between 0 and 11), then there are over 90 million possible combinations of feature values. Yet the dictionary used here only accounts for 6074 of those combinations, less than 0.0065%. This means that the likelihood of a mutation producing an invalid set of feature values is prohibitively high. An iterative procedure like the simplistic one mentioned above could take a long time to run. Because this evolutionary algorithm operates in a real-time environment, where the customers are (impatient) children, minimizing run-time is vital. A target maximum of 1 second was chosen for the algorithm to run in its entirety.

One approach to the sparse feature space problem would be conceptually to mutate from one entry in the domain to another, rather than from one vector to another. As indicated by figure 8.1, all words in the domain can be represented as points in 7-dimensional space, thus it is possible to sort the entire dictionary according to the entries' feature values. This would mean computing a 35000×35000^2 matrix containing the distance in feature space from each entry to every other entry. Then when making mutations, the algorithm need only look up entries in this matrix — small mutations would look for close neighbors and large

²The size of the database is approximately 35000 entries. Since the distance between any two entries is symmetrical, the size of the matrix really need only be $(35000 \times 35000)/2$.

mutations would look further away. However, again, practical considerations render this solution infeasible because too much memory is required to store this matrix³.

An alternative to storing the entire table in memory would be to load the relevant portion from disk during run-time; however testing proved that selective loads took longer than the 1 second time requirement. Another option would be to compute the relevant portion of the table during run-time; again, testing showed that this method exceeded the maximum time requirement.

The final solution was to adopt a new reproduction process called *reproduction through sampling*. The strategy is to begin by randomly selecting a relatively small sample population from the dictionary and then to replace the parents in G''' with children chosen from this sample. An overview is shown in figure 8.4 and the details are in table 8.5.

³“Too much” simply means more than is available on the CEL server.

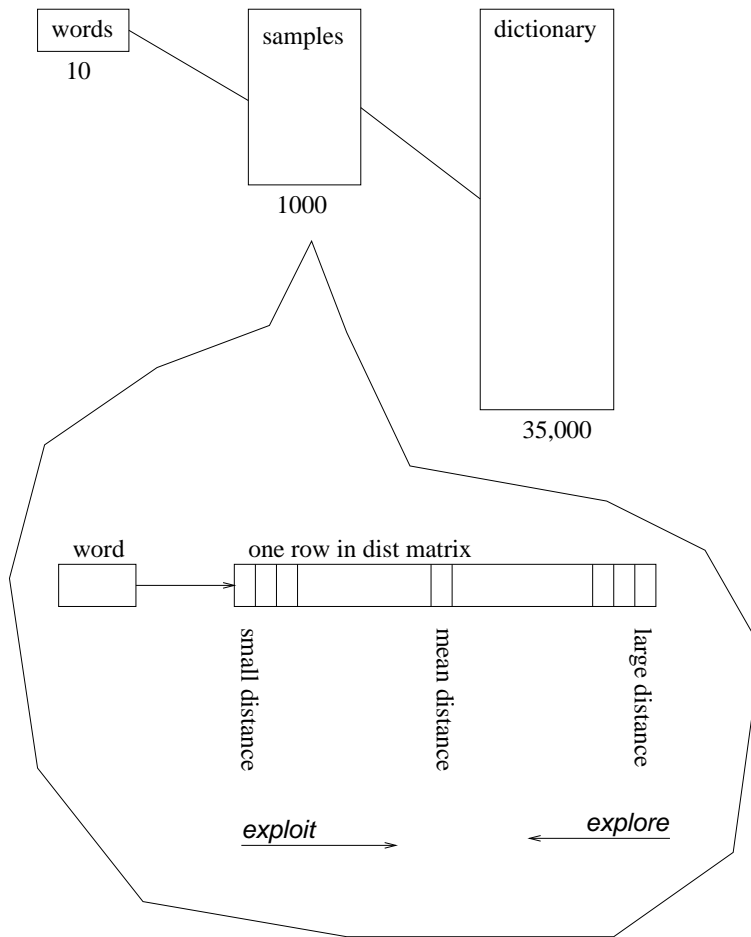


Figure 8.4: Sampling illustration.

Table 8.5: Reproduction algorithm.

```

1. Select a random sample of 1000 words from the 35,000 word dictionary:
   for i ← 1 to 1000
     x ← random( 1, 35000 );
     Si ← dictionaryx;

2. Compute the distance between all entries in S and G''':
   for j ← 1 to 1000
     for i ← 1 to ng'''
       disti,j ← [ (∑k=1ng''' [(gi,k''' - sj,k) / (maxk - mink)]2) * k ];

3. Sort each row in the distance matrix:
   for i ← 1 to ng'''
     sort( disti );

4. Compute the mean for each row in the distance matrix:
   for i ← 1 to ng'''
     meandisti ← mean( disti );

5. For each row in the distance matrix, save the index of the value closest
   to the mean:
   for i ← 1 to ng'''
     xmeandisti ← index of entry in disti whose value is closest
                   to meandisti;

6. Generate a child for each parent in G''':
   for i ← 1 to ng'''
     if gi''' has a score
       if ( gi''' .score < avg ) then
         j ← pickDist( i, FAR ); /* explore */
       else
         j ← pickDist( i, NEAR ); /* exploit */
     else
       j ← pickRandom(); /* new word */
     gi''' = sj;

```


An example is shown in table 8.6, illustrating the relationship between one parent and one child word list. This data was taken from the data set of one of the students involved in the pilot study.

Table 8.6: Distance between words in successive generations.

	parent		child		dist
1	four	[4,3, 7,2,2,0,0]	four	[4,3, 7,2,2,0,0]	0
2	who	[3,5, 9,1,2,1,0]	aim	[3,8, 5,2,1,0,0]	1
3	race	[4,7, 6,2,2,0,0]	peas	[4,6, 6,2,2,0,0]	1
4	vies	[4,7, 7,2,2,0,0]	dates	[5,4, 6,2,3,0,0]	2
5	away	[4,5,10,2,2,0,0]	fives	[5,7,11,2,3,0,0]	2
6	singed	[6,9, 8,2,4,1,0]	calorie	[7,7, 9,4,3,0,0]	2
7	forked	[6,3,14,2,4,1,0]	debated	[7,8,11,3,4,0,0]	3
8	enumerates	[10,9,12,5,5,0,0]	ragged	[6,3, 9,2,4,1,0]	2
9	manipulated	[11,9,16,5,6,0,0]	perused	[7,6,10,3,4,0,0]	1
10	fosters	[7,4,10,2,5,2,0]	numerics	[8,9,12,3,5,1,0]	3

8.3 Results

Data collected in the pilot study described in chapter 6, from the games of Keyit and Pickey, were used for the analysis here. The domain coverage for each user was examined, to determine if the evolutionary approach to word selection led players into more of the domain space than a pre-leveled application might. Additionally, the relationship between typing speed and various word features was analyzed, to determine which features, if any, emerged as more highly correlated (to typing speed) than others.

8.3.1 Domain coverage

The seven feature values that define the domain are: word length, Scrabble score, keyboarding level, number of vowels, number of consonants, number of 2-character consonant clusters, number of 3-character consonant clusters. (These are described in detail in chapter 5.) Scrabble score and keyboarding level are used for analysis here, since the remainder are a function of word length (as is Scrabble score) and so can be considered redundant in this analysis.

The chart in figure 8.5 is a sample domain coverage chart, plotting Scrabble score versus keyboarding level. A point exists in the domain space for each circle on the plot. For each point that a user has been exposed to, the circle is filled (\bullet). Thus the open circles (\circ) represent portions of the domain space that the user has not seen. This sample chart illustrates the coverage that a user might

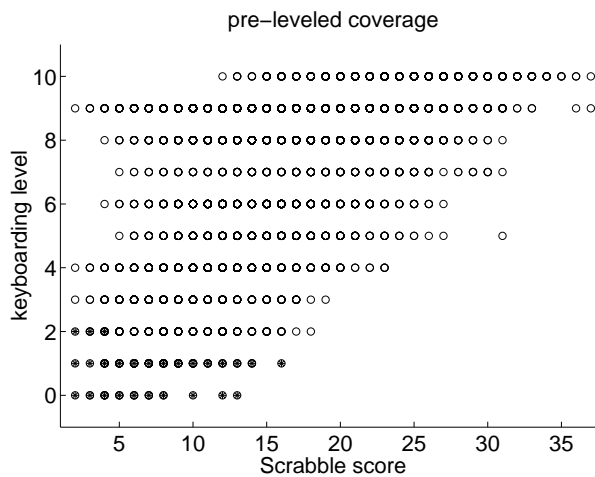


Figure 8.5: Sample domain coverage chart.

experience in a pre-leveled environment, where (e.g.) she must complete all problems in keyboarding levels 0 and 1 before seeing any problems from level 2.

Figure 8.6 contains domain coverage charts for the four sample students from the pilot study (see chapter 6). All the students have been exposed to a large portion of the domain. Students 88 and 89, who are faster typers than the other two, have seen more of the domain. The slowest typer, 119, has more concentrated domain coverage.

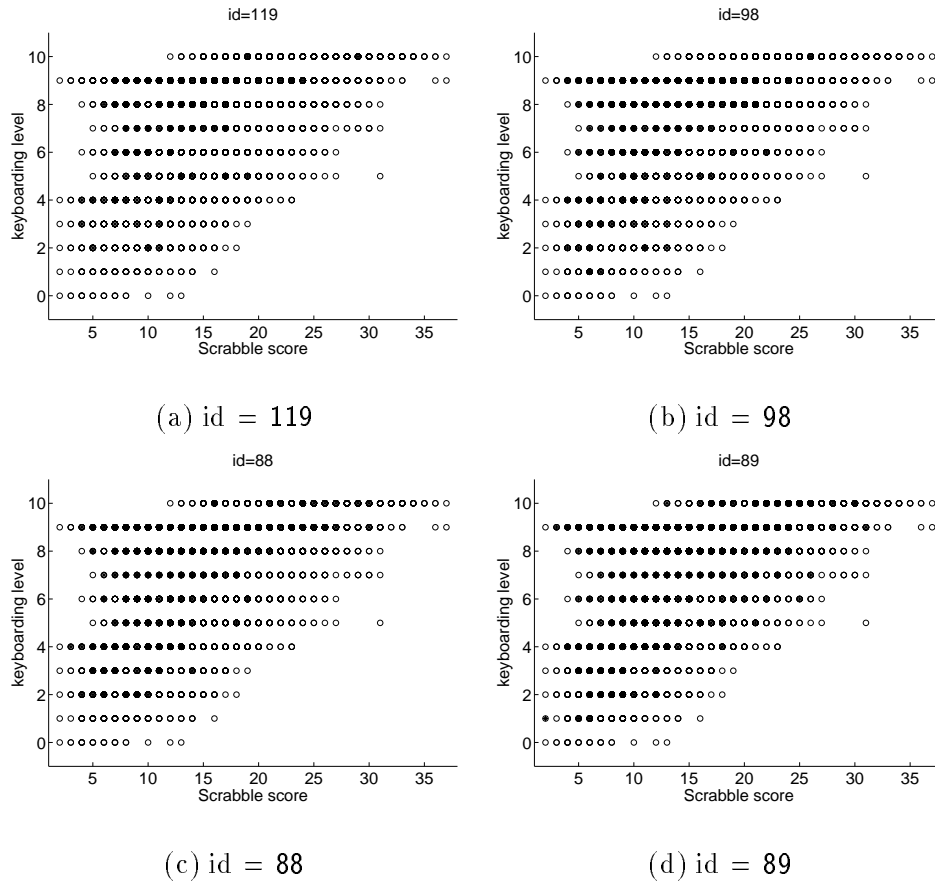
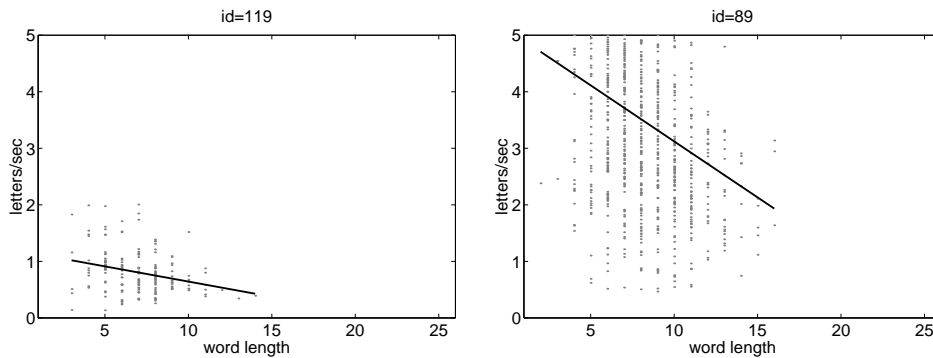


Figure 8.6: Domain coverage charts for sample users.

8.3.2 Feature correlation

The relationships between word length, Scrabble score, keyboarding level and typing speed are examined here, in order to ascertain if any one of these three features appears to correlate more closely with typing speed than any of the others. There should be a direct correlation between word length and typing speed. It has been demonstrated that an artifact of typing long words exists such that the speed per letter is slower than for shorter words [Larochelle, 1982]. Thus, even when the time it takes to type a word is normalized for the length of the word, so that speed is measured in letters per second, longer words still take more time to type than shorter words.

Figure 8.7 shows plots for two of the sample students. On each graph, there is a point for each word typed by the corresponding student. The straight line is a linear least-squares fit of all the points. The artifact (i.e., longer words take longer to type) is readily apparent in the figure.



(a) id = 119

(b) id = 89

Figure 8.7: Word length vs typing speed.

Figure 8.8(a) contains the fitted lines for all of the students involved in the pilot study. The lines for the four students shown in figure 8.7 are highlighted. The same behavior pattern, where per letter typing speed is reduced (slower) for longer words, is consistent with every user. Figures 8.8(b) and 8.8(c)⁴ show the relationships between Scrabble score and keyboarding level with typing speed, respectively.

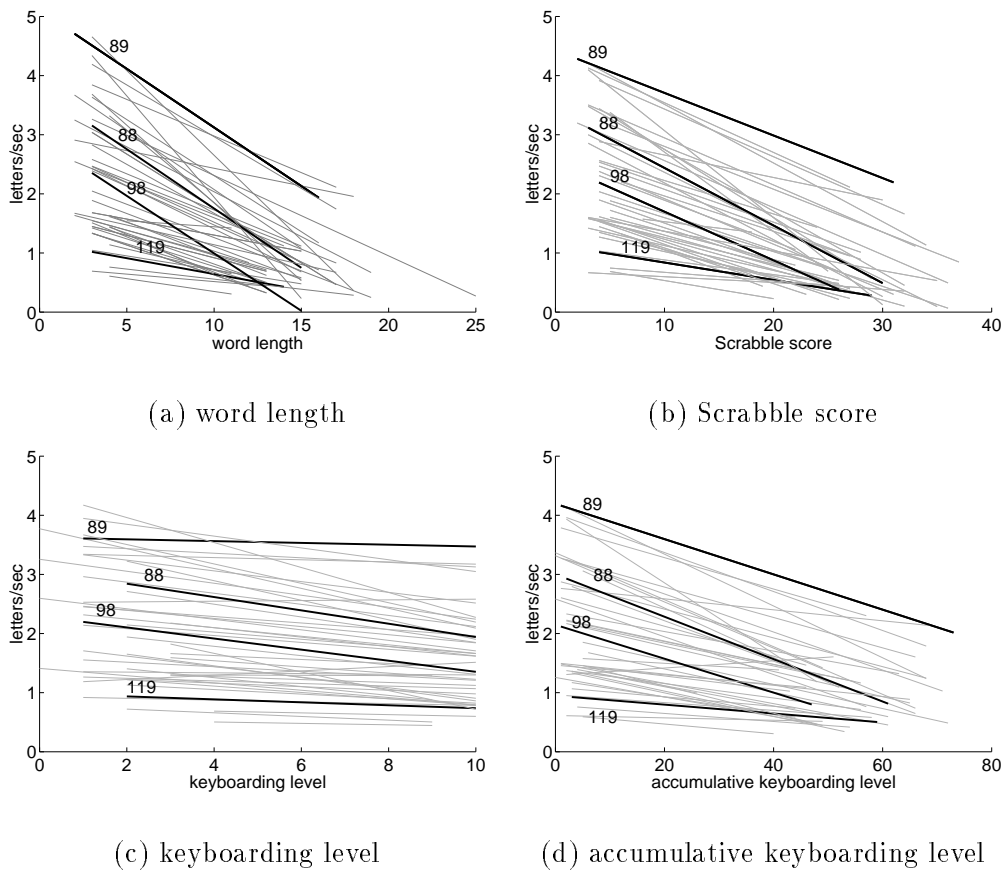


Figure 8.8: Feature correlation with typing speed.

⁴The accumulative keyboarding level shown in figure 8.8(d) is explained ahead.

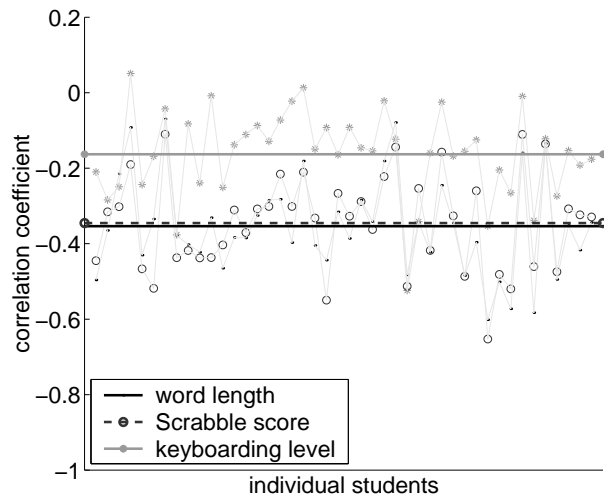
The preceding plots (8.8(a) through 8.8(c)) appear to indicate that typing speed correlates more directly with word length and Scrabble score than with keyboarding level. Figure 8.9(a) shows the correlation coefficients, for each user, between typing speed and each of these features. Each point on the chart corresponds to one of these statistics per user (i.e., there are three points per user). Faint lines connect the points for each feature, to make it easier for the reader to group the points. Horizontal lines are drawn to indicate the mean correlation coefficient for each feature, across all users. A correlation coefficient closer to -1 indicates a higher negative correlation between two variables — e.g., that longer word length is indicative of slower typing speed, per letter.

Scrabble score is a function of word length, since each letter in the word contributes to the score individually. Conversely, keyboarding level is computed independently of word length. So it is not surprising that word length and Scrabble score exhibit similar statistical characteristics. Indeed, the higher correlation for word length dependent statistics and typing speed confirm the statement made in the previous section: that long words take more time to type, on a per letter basis, than short words.

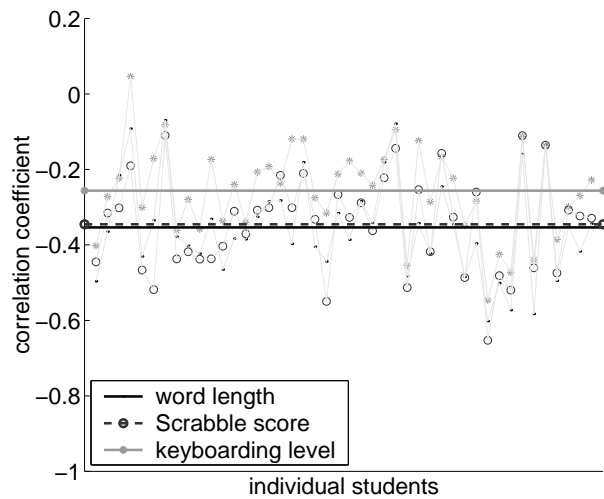
For comparison, a modified keyboarding level was computed in which keyboarding level is also defined as a function of the length of the word (as is Scrabble score and obviously word length). Instead of calculating keyboarding level to be chosen as the highest level of any letter in a word, the levels of all the letters in the word were totalled (the same way that Scrabble score is computed). This *accumulative keyboarding level* was computed for all the data, after the pilot study

was finished. The purpose was to determine if keyboarding level really correlated so poorly to typing speed as was indicated in figure 8.9(a), or if the method of computing keyboarding level (where word length was not a factor) skewed the correlation results away from keyboarding level. After all, since the data was collected during keyboarding games, it would seem logical that keyboarding level should be highly correlated to typing speed.

The correlation results, with the accumulative keyboarding levels, are shown in figures 8.8(d) and 8.9(b). It was found that the correlation coefficient for accumulative keyboarding level is -0.256054 , a better correlation than that with the original keyboarding level (-0.163075). However, word length and Scrabble score still correlate significantly higher than keyboarding level.



(a) typing speed vs word length, Scrabble score and keyboarding level



(b) typing speed vs word length, Scrabble score and accumulative keyboarding level

	mean	std. deviation	median
word length	-0.353444	0.130214	-0.371364
Scrabble score	-0.345417	0.127598	-0.326716
keyboarding level	-0.163075	0.116377	-0.153707
accumulative keyboarding level	-0.256054	0.119275	-0.240934

Figure 8.9: Correlation coefficients.

8.4 Discussion

The primary advantages of using an evolutionary approach for guiding problem selection in an educational game include:

- Students guide themselves through the domain, based on their own performance with the system, which means that students whose learning patterns are non-standard may benefit.
- Students may reach areas of the domain that they may not see otherwise, where a standard pre-leveled system may prevent them from leaving an area without successfully completing all the problems in that area. This could be seen as providing an uncertain goal, one of Malone's pointers for helping to motivate learners in an educational game.
- Costs and effort to implement the game are reduced, because the domain need not be analyzed at such a fine grain level as engineered systems require.

The fact that word length and Scrabble score correlate to typing speed much more directly than keyboarding level (both original and accumulative) is a result which bears further study. The indication is that future words should be chosen more on a basis of word length and Scrabble score than keyboarding level.

It has been suggested that frequency of word usage in the English language should also be a feature in the domain space for word games⁵. Although this is partially encoded in the frequency of letter count that is part of Scrabble

⁵Suggested by Andrew Howard.

score, use of a precise statistic may be beneficial. While touch typing courses commonly require learners to type non-linguistic sequences of characters, people will generally say that they can type words they know faster than words they don't know. Future work will involve adding this dimension to the feature space and studying the resulting correlations.

The results shown in this chapter indicate that the use of an evolutionary approach is a viable alternative to pre-leveled methods. In future work, control studies must be performed in order to validate this statement. For example, some students' games would be supplied with words chosen by the method described here, others would be chosen at random and others would be chosen according to a standard pre-leveled curriculum.

The flexible design of the CEL system permits studies like this one to be enabled easily. The modular "plug and play" architecture allows researchers to substitute different game content selection engines without needing to build an entire system. The benefit of using an evolutionary approach like this inside CEL is that patterns of usage may emerge that could not otherwise be revealed in a highly-engineered system or a system that did not have access to the large number of users that an Internet system offers.

Chapter 9

Conclusion

This thesis has described the CEL system which was built to enable an Internet learning community that can support the types of activities, experiments and data collection common to the ILS field. The work presented addresses three specific needs that have previously not been met satisfactorily by other interactive learning systems: accessibility, flexibility and extensibility. This concluding chapter begins by defending these three claims. Then we discuss CEL in relation to current Internet community issues and close with an outline of future work.

9.1 Accessibility

We have demonstrated that CEL is accessible to Internet users, first by testing on a variety of platforms and then through a pilot study conducted at a public primary school. The client software used by participants runs inside a standard Internet browser and is compliant with Netscape 3.0 (or higher), which is more

widely available than later versions. No additional software is required on the client site, which means that schools do not have to shoulder the burden of keeping up with special downloads and upgrades in order to provide students with access to CEL.

The CEL client is designed to operate on computers with limited memory and low network bandwidth. Java applets that are part of the client are kept small, which serves not only to prevent participants from running out of memory on their computers but also minimizes the amount of time it takes for applets to load. Socket connections which facilitate communication between clients' computers and the CEL server are kept open for short time periods, preventing the type of failures that occur when longterm connections are required.

9.2 Flexibility

We have demonstrated CEL as a flexible platform that can house a variety of interactive learning activities. CEL can support multi-player collaborative or competitive, real-time or turn-taking activities. All activities currently built for CEL are two-player educational games. We have described examples of each of these in chapter 3. Keyit and Pickey are competitive, asynchronous, two-player typing games. Automath is a competitive, asynchronous, two-player arithmetic game. Monkey is a collaborative, asynchronous, two-player word game. Loois is a collaborative, turn-taking, two-player construction game. Tron is a competitive, real-time two-player spatial reasoning game. The first three games are based on

a traditional drill-and-practice format. The fourth and fifth games are based on constructionism. All are designed to support learner-centered learning.

9.3 Extensibility

We have demonstrated that CEL puts forth an easily extensible model. In building the system, we first created and tested the game of Keyit. All subsequent games were implemented by extending the four base Java classes created for Keyit: `PlaygroundGame`, `Matchmaker`, `MatchmakerThread` and `SecretAgent`. This thesis contains brief descriptions of each these classes. Full software documentation can be found in [Sklar, 2000], which in future will be available on our web site, along with the Java class files, so that others can download the classes and contribute their own activities to the CEL community. The extensible nature of CEL saves teachers and researchers from needing to build an entire multi-user system from scratch.

9.4 Issues in Internet communities

Throughout this thesis, we have touted accessibility as a desirable feature because (1) researchers can reach a potentially very large number of subjects, (2) teachers can share activities, and (3) students can interact and learn from each other without needing to be in the same physical location. However, there are other aspects of accessibility that are less favorable, namely safety and privacy. This section discusses these concerns as they relate to CEL. First we review the issues

of Internet safety and privacy. Then we explain how the environment works to protect its users, by providing an alternate method for identity (through IDsigns) and facilitating only indirect communication between participants.

9.4.1 Safety and privacy in CEL

Unrestricted communication on the Internet is a worry for parents, who are concerned about whom their children are communicating with and what kind of information their children are revealing about themselves — both actively and passively. In June 1998, the Federal Trade Commission (FTC) reported that a majority of sites failed to tell visitors how they used the personal information that they were collecting. Typically, sites did not ask for children to obtain permission before providing information. “The commission now recommends that Congress develop legislation placing parents in control of the online collection and use of personal information from their children.” [CNN, 1998]

Over the last two years, since that report was issued, many sites have added shrink-wrapped consent forms (similar to the form employed in CEL), as well as on-line explanations of how the information collected from participants is being used. Most of these sites, including ours, ask children under age 18 to obtain permission from a parent or guardian before joining an on-line community. However, there is no fool-proof way to ensure that children are actually getting this permission. Some sites ask for new users to enter a credit card number, under the assumption that children would not have access to such information without parental involvement — but this assumption is weak. Creating and providing a

fictitious identity is extremely easy to do and attractive to many. Beth Givens, who runs Privacy Rights Clearing House in California, says: “A lot of people give false information, and quite proudly. It becomes a game.” [CNN, 1998]

Even though everyone in a community may be using an alias and a fictitious identity, an anonymous exchange of inappropriate ideas or language may still occur. On sites that allow open chat, there is no protection to prevent users from exchanging personal information such as addresses, phone numbers, etc. Some educational sites geared towards children employ an adult user to monitor all communication, censoring inappropriate material and ensuring that the students stay “on task”. But this approach is not practical in a real-time Internet system, where participation occurs on a 24-hour basis. Providing a safe environment in which young students can interact is of primary concern. In CEL, all communication is accomplished through the moves of the games participants are playing; no direct chat is implemented, and thus all users are safe.

Privacy is another issue of concern. Many sites collect data by observing mouse clicks and do not ask for explicit permission from their users. Amazon¹ gathers statistics on the products that their customers order and then uses this data to present recommendations. Their handling of the privacy issue is typical. On their home page, they provide a link to a separate page that contains their privacy policy, stating:

¹<http://www.amazon.com>

By using our Web site, you consent to the collection and use of this information by Amazon.com. If we decide to change our privacy policy, we will post those changes on this page so that you are always aware of what information we collect, how we use it, and under what circumstances we disclose it. [Amazon, 2000]

Further, they state that they reserve the right to sell this information to “trustworthy third parties”, but you may send them email requesting that data collected under your account is never sold.

In CEL, our login screen contains a warning, which appears every time a user logs in, not just the first time, and it appears in full at the bottom of the screen, without requiring users to click to another, out-of-sequence page, in order to see the text:

By logging in, I consent to being involved in this experiment using the pseudonymous name that I have entered above. I understand that data is collected on every game I play, and I hereby give the DEMO Lab at Brandeis University permission to analyze and publish this data for scientific purposes.

For our pilot study, we provided parents with information about CEL and our experiments. In turn, the parents gave us written permission to use their children’s input.

The work presented in this thesis specifically advocates the use of the Internet as a virtual laboratory in which to collect data from thousands of users. There is an inherent conflict between privacy and need for input in any study involving humans. Proper treatment of information gathered on the Internet is warranted, and this data should be handled according to standard practices, such as those in place at any psychology laboratory or hospital.

9.4.2 Identity in CEL

As a means toward protecting user privacy and providing anonymity, CEL implements an alternative method for identifying users. The most common form of user identification in virtual communities is a username that individuals create for themselves when logging onto a site for the first time. A few on-line communities provide graphical icons with which users identify themselves to others. Generally (if not exclusively, aside from CEL), these graphics have already been created and users select pictures from a list and then designate the pictures to represent themselves. On some sites, the system assigns the picture without any input from the user. At least one protective children's site lets children choose text user names from a list and then designate the names to represent themselves.

In CEL, users are identified by IDsigns (see chapter 3). In the pilot study outlined in chapter 6, we found that the children were extremely creative with their IDsigns and that this element of the system was very popular. A wide variety of IDsigns were created, some of which are shown in figure 9.1.



Figure 9.1: Sample IDsigns.

As we suspected would happen, the children quickly discovered that they could make short words inside their IDsigns. Upon this occurrence, they were cautioned not to use their real names. We found it quite amusing when one boy created an IDsign that had one of his classmates' first names in it, which led many children to believe they were playing games with the classmate.

We found it interesting to observe that there were no discernible differences between age or gender groups in the types of designs created. Boys and girls of all ages (those in the sample, and even adult participants) were equally likely to create patterns, faces or figures. Indeed, the heart design shown in figure 9.1 — a stereotypical “girl drawing” — was created by a boy.

In future work, we may conduct a study on identity in CEL. We can follow the trends of single users, examining how (and if) they change their IDsigns over a period of time and across a number of sessions with CEL. We can look for trends in the population of CEL users, for example, design features that might align with demographics, such as age, gender or location.

9.4.3 Communication in CEL

Another feature of CEL that works to protect users is our indirect method of facilitating communication between participants. This is in contrast to most Internet communities where direct communication happens openly in chat rooms. Instead, CEL participants interact with each other and/or with software agents, through a limited language — i.e., the moves of the games they play. This restricted mode of communication serves two purposes. First, it protects the

privacy of young participants, as discussed in the preceding paragraphs. Second, the software agents that were described in chapter 7 interact successfully in CEL because the hurdle of natural language processing that normally accompanies the task of building an agent to act as a human is avoided. In the CEL system, the computer becomes a mediator, both actively in the form of agents as artificial playmates, and passively in the form of the server passing interactions from one human player to another.

In future work, we plan to explore the concept of non-conversational collaborative learning. This idea diverges from typical computer supported collaborative learning (CSCL) systems, where open conversation is permitted and advocated. However, the CEL environment may facilitate learning partnerships that might not occur in a conversational setting, perhaps between two children who do not speak the same language.

9.5 Future work

Aside from the projects mentioned earlier in this chapter, current and future work with the CEL system falls into three areas:

1. enlarging the game set,
2. improving site visualization, and
3. matching playmates appropriately.

Much of the future with CEL relies on enlarging the game set and involving researchers who work in the areas of education, psychology and cognitive sci-

ence. Currently, we are building an arithmetic game that is more complex than Automath, a spelling bee and a geography quiz.

9.5.1 Visualization

In CEL, visualizing who is logged into the community is done through the IDsigns and the playground. This mechanism aligns with the requirements for web site visualization laid out in [Minar & Donath, 1999]: individuals are represented and the display is animated. Unlike Minar and Donath’s work, the visual display in CEL is updated in pseudo real-time, because users need to know who is logged in to which game at any given time.

Currently, there is no spatial structure on the site — i.e., relating one game to another. Future work will examine defining each game as a room and giving each user an omniscient vantage point. On the games menu page (figure 3.3), users will be able to see who is logged into any game and thus select which playground to enter based on who is already there or how many other players are present. This type of viewpoint is given on games sites like Yahoo² and Yahoo!igans³.

9.5.2 Player clustering

The most significant next project that we have planned for CEL is to implement a matching algorithm that builds playgroups based on appropriate membership — not just according to who is logged on at any given moment, which is the

²<http://games.yahoo.com>

³<http://www.yahooligans.com>

method used now. The idea is to form playgroups for each student such that the playmates therein can provide appropriate challenges to motivate learning.

Playmates will be assigned based on users' student models and participants will only be able to "see" playmates who are "appropriate". This notion of restricting players' views to a subset of playmates distinguishes CEL from other game playing web sites. Typically, users can see everyone else who is currently logged in; this is the method used at the Yahoo games site, for example.

Of course, given the unpredictability of human behavior, it will be impossible to surmise precisely what a potential playmate's behavior will be, but an educated guess can be made. A record of predictions and outcomes will be maintained in order to obtain a measure of the reliability of the predictive mechanism, as it applies to each user. Taking this prediction reliability rating and each user's performance profile as input, users will be clustered and playgroups will be formed. These groupings are highly dynamic, as users enter and exit the system and as games are played and the performance profile and prediction ratings change.

A playgroup can be represented conceptually as an undirected graph, where each player is a vertex in the graph. Edges are drawn between players who are considered to be appropriate playmates. Edges are updated as players enter and exit games, as games are played and as users progress.

An example is shown in figure 9.2. A player only sees those players that are in his playgroup. In the example shown, this means that even though players 1 and 2 are both connected to the same game at the same time, they do not see each other's IDsigns in their playgrounds because the system does not deem

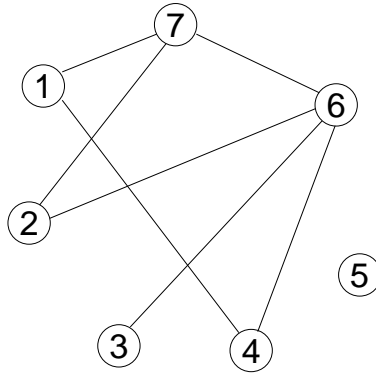


Figure 9.2: Sample playgroup graph.

Player 1's playgroup contains mates 4 and 7; player 2's playgroup contains mates 6 and 7; player 3's playgroup contains mate 6; player 4's playgroup contains mates 1 and 6; player 5 has no playmates; player 6's playgroup contains mates 2, 3, 4 and 7; player 7's playgroup contains mates 1, 2 and 6.

them to be appropriate playmates. Connections are bi-directional, so (e.g.) if player 1 sees player 7, then player 7 also sees player 1. However, links are not transitive: players 1 and 7 see each other, players 7 and 2 see each other, but players 1 and 2 do not.

Two algorithms will be explored for defining playgroups, (which is equivalent to determining the edges of the graph in figure 9.2). Each uses a different approach, one *absolute* and one *relative*. In the first approach, all players are ranked according to an absolute scale and only vertices of players whose ranks are within a certain epsilon (ϵ) of each other are connected. This algorithm maintains an auxiliary index on the list of players, sorted by rank. Edges are drawn on the graph by sliding a window of width $2 * \epsilon$ down the indexed list of players and connecting all players that are inside the window. The second method computes

a relative distance between every pair of players in the graph and only connects vertices of players whose distances are within a certain epsilon of each other.

There are two reasons to favor use of the second method. First, relative matching may produce more accurate outcomes in terms of shared experiences that are beneficial to both players. Second, in some domains, it may be difficult to define an absolute ranking.

In addition to these methodologies, known clustering algorithms will be applied: Cobweb [Fisher, 1987], Unimem [Lebowitz, 1987], c4.5 [Quinlan, 1993], MML [Wallace, 1990], LSI [Deerwester et al., 1990] and other statistical methods. A comparison of the performance of these algorithms, measured in terms of run-time and effectiveness of output, will determine the method that works best for CEL.

This work will also be expanded to include software agents. If the above calculations are performed and it is deemed that no appropriate human partners are currently logged into CEL, then the system can select an appropriate human partner from those who are *not* logged into the system and instantiate an appropriate software agent, trained on that human's performance data (as described in chapter 7).

9.6 Finally

Comments from the teachers who participated in our pilot study were quite positive. They felt that their students had become stronger typers, and they are enthusiastic about expanding CEL to cover new domains. “CEL is easily configured for the material and skills I want to reinforce,” one teacher said.

The computer gaming environment seems naturally competitive, and we were concerned that the competitive aspects of the environment might be perceived negatively by educators, however, one stated: “It’s the best way I’ve seen students compete academically without causing a lot of problems.” An additional remark was a feature we had not considered: “Their experience in non-threatening test-taking was increased.”

CEL has been shown to have the flexibility to host a variety of different types of interactive activities. Since CEL resides on the Internet, researchers can use CEL to collect data from a very large population of subjects with varying ages, genders, abilities and locations. This experimental setting contrasts with that of most interactive learning system studies, which typically involve use of one particular activity, implemented in a controlled setting and accessed by a limited number of homogeneous subjects. The extensible design of CEL positions the system well for future contributions from others interested in experimenting with this unique type of safe, fun and interactive environment for children.

Bibliography

- [Amazon, 2000] Amazon (2000). <http://www.amazon.com/exec/obidos/subst/misc/policy/privacy.html/002-0198712-6193016>.
- [Amory et al., 1998] Amory, A., Naicker, K., Vincent, J., & Adams, C. (1998). Computer Games as a Learning Resource. In *Proceedings of the World Conference on Educational Multimedia, Hypermedia & Telecommunications (Ed-Media98)*.
- [Anderson, 1982] Anderson, J. R. (1982). Acquisition of Cognitive Skill. *Psychology Review*, 89.
- [Anderson, 1993] Anderson, J. R. (1993). *Rules of the Mind*. Lawrence Erlbaum Associates, Hillsdale, NJ.
- [Anderson et al., 1995] Anderson, J. R., Corbett, A. T., Koedinger, K., & Pelletier, R. (1995). Cognitive tutors: Lessons learned. *The Journal of Learning Sciences*, 4:167–207.
- [Beck, 1997] Beck, J. (1997). Modeling the Student with Reinforcement Learning. In *Proceedings of the Machine Learning for User Modeling Workshop, Sixth International Conference on User Modeling*.
- [Bell et al., 1995] Bell, P., Davis, E. A., & Linn, M. C. (1995). The Knowledge Integration Environment: Theory and Design. In *Proceedings of Computer Supported Collaborative Learning (CSCL'95)*.
- [Brody, 1993] Brody, H. (1993). Video Games That Teach? *Technology Review*, November/December.
- [Brown & Burton, 1978] Brown, J. S. & Burton, R. B. (1978). Diagnostic Models for Procedural Bugs in Basic Mathematical Skills. *Cognitive Science*, 2(2).
- [Bruckman, 1997] Bruckman, A. (1997). *MOOSE Crossing: Construction, Community, and Learning in a Networked Virtual Community for Kids*. PhD thesis, MIT.

- [Bruckman & DeBonte, 1997] Bruckman, A. & DeBonte, A. (1997). MOOSE Goes to School: A Comparison of Three Classrooms Using a CACL Environment. In *Proceedings of CSCL'97*.
- [Brusilovsky et al., 1996] Brusilovsky, P., Schwarz, E., & Weber, G. (1996). ELM-ART: An Intelligent Tutoring System on World Wide Web. In Frasson, C., Gauthier, G., & Lesgold, A., editors, *Intelligent Tutoring Systems (Lecture Notes in Computer Science, Vol. 1086)*, pages 261–269. Springer Verlag.
- [Cherry, 1978] Cherry, C. (1978). *On Human Communication*. MIT Press, Cambridge, 3rd edition edition.
- [Clancey, 1986] Clancey, W. J. (1986). Intelligent Tutoring Systems: A Tutorial Survey. Technical Report STAN-CS-87-1174, Stanford University.
- [CNN, 1998] CNN (4 June 1998).
- [Conati & VanLehn, 1996] Conati, C. & VanLehn, K. (1996). POLA: a student modeling framework for Probabilistic On-Line Assessment of problem solving performance. In *Proceedings of the Fifth International Conference on User Modeling (UM-96)*.
- [Dede et al., 1996] Dede, C., Salzman, M., & Loftin, B. (1996). ScienceSpace: virtual realities for learning complex and abstract scientific concepts. In *Proceedings of IEEE Virtual Reality Annual International Symposium*.
- [Deerwester et al., 1990] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, 41(6):391–407.
- [Duncan et al., 1990] Duncan, C. H., VanHuss, S. H., Warner, S. E., & O'Neil, S. L. (1990). *College keyboarding/typewriting : complete course*. South-Western Publishing Co., Cincinnati, OH.
- [Fanderclai, 1995] Fanderclai, T. (1995). MUDs in Education: New Environments, New Pedagogies. *Computer-Mediated Communication Magazine*, 2(1).
- [Fisher, 1987] Fisher, D. (1987). Knowledge Acquisition via Incremental Conceptual Clustering. *Machine Learning*, 2:139–172.
- [Forrester, 1992] Forrester, J. (1992). System Dynamics and Learner-Centered-Learning in Kindergarten through 12th Grade Education. Technical Report D-4337, MIT.
- [Funes & Pollack, 1998a] Funes, P. & Pollack, J. B. (1998a). Componential Structural Simulator. Department of Computer Science Technical Report CS-98-198, Brandeis University.

- [Funes & Pollack, 1998b] Funes, P. & Pollack, J. B. (1998b). Evolutionary Body Building: Adaptive physical designs for robots. *Artificial Life*, 4:337–357.
- [Funes et al., 1997] Funes, P., Sklar, E., Juillé, H., & Pollack, J. B. (1997). The Internet as a Virtual Ecology: Coevolutionary Arms Races Between Human and Artificial Populations. Department of Computer Science Technical Report CS-97-197, Brandeis University.
- [Funes et al., 1998] Funes, P., Sklar, E., Juillé, H., & Pollack, J. B. (1998). Animal-Animat Coevolution: Using the Animal Population as Fitness Function. In *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*.
- [Gonzalez, 2000] Gonzalez, A. (April 26, 2000). Digital divide closes — but schools aren't ready. *USA Today*.
- [Gordin et al., 1996] Gordin, D. N., Gomez, L. M., Pea, R. D., & Fishman, B. J. (1996). Using the World Wide Web to Build Learning Communities in K-12. *The Journal of Computer-Mediated Communication*, 2(3).
- [Gordon & Hall, 1998] Gordon, A. & Hall, L. (1998). Collaboration with Agents in a Virtual World. In *Workshop on Current Trends and Applications of Artificial Intelligence in Education: 4th World Congress on Expert Systems*.
- [Gruber & Voneche, 1977] Gruber, H. E. & Voneche, J. J., editors (1977). *The Essential Piaget*. BasicBooks.
- [Hasbro, 1999] Hasbro (1999). Scrabble 101.
- [Healy, 1999] Healy, J. M. (1999). *Failure to Connect: How Computers Affect Our Children's Minds — and What We Can Do About It*. Touchstone Books.
- [Holland, 1975] Holland, J. H. (1975). *Adaption in Natural and Artificial Systems*. University of Michigan Press.
- [Hughes, 1995] Hughes, B. (1995). Educational MUDs: Issues and Challenges.
- [Johnson & Johnson, 1989] Johnson, D. W. & Johnson, R. (1989). Cooperative Learning, Values, and Culturally Plural Classrooms. In *Cooperation and competition: Theory and research*. Interaction Book Company.
- [Kinshuk & Patel, 1997] Kinshuk & Patel, A. (1997). A Conceptual Framework for Internet based Intelligent Tutoring Systems. *Knowledge Transfer*, II.
- [Klawe & Phillips, 1995] Klawe, M. & Phillips, E. (1995). A Classroom Study: Electronic Games Engage Children as Researchers. In *Proceedings of Computer Supported Collaborative Work (CSCL'95)*.

- [Klawe et al., 1996] Klawe, M., Westrom, M., Davidson, K., & Super, D. (1996). Phoenix Quest: lessons in developing an educational computer game for girls ... and boys. In *Proceedings of ICMTM96*.
- [Koedinger & Anderson, 1993] Koedinger, K. & Anderson, J. (1993). Effective use of intelligent software in high school math classrooms. In *Proceedings of the World Conference on Artificial Intelligence in Education*.
- [Kohn, 1986] Kohn, A. (1986). *No Contest: The case against competition*. Houghton-Mifflin.
- [Kolodner, 1983] Kolodner, J. L. (1983). Maintaining organization in a dynamic long-term memory. *Cognitive Science*, 7.
- [Koza, 1992] Koza, J. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA.
- [Larochelle, 1982] Larochelle, S. (1982). A Comparison of Skilled and Novice Performance in Discontinuous Typing. In Cooper, W., editor, *Cognitive Aspects of Skilled Typewriting*, pages 67–94, New York. Springer-Verlag.
- [Lebowitz, 1987] Lebowitz, M. (1987). Experiments with Incremental Concept Formation: UNIMEM. *Machine Learning*, 2:103–138.
- [Littman & Soloway, 1988] Littman, D. & Soloway, E. (1988). Evaluating ITSs: the cognitive science perspective. In Polson, M. C. & Richardson, J. J., editors, *Foundations of Intelligent Tutoring Systems*. Lawrence Erlbaum Associates, Hillsdale, NJ.
- [Mark & Greer, 1993] Mark, M. A. & Greer, J. E. (1993). Evaluation Methodologies for Intelligent Tutoring Systems. *Journal of Artificial Intelligence and Education*, 4:129–153.
- [McCalla & Greer, 1994] McCalla, G. I. & Greer, J. E. (1994). Granularity-Based Reasoning and Belief Revision in Student Models. In *Student Models: The Key to Individualized Educational Systems*, pages 39–62, New York. Springer Verlag.
- [McGrenere, 1996] McGrenere, J. L. (1996). Design: Educational Electronic Multi-Player Games; A Literature Review. Department of Computer Science Technical Report 96-12, University of British Columbia.
- [Minar & Donath, 1999] Minar, N. & Donath, J. (1999). Visualizing the Crowds at a Web Site. In *Proceedings of CHI'99*.
- [Papert, 1980] Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. BasicBooks.

- [Papert, 1991] Papert, S. (1991). Situating Constructionism. *Constructionism*.
- [Papert, 1993] Papert, S. (1993). *The Children's Machine*. BasicBooks.
- [Pea, 1993] Pea, R. (1993). The collaborative visualization project. *Communications of the ACM*, 36(5):60–63.
- [Pomerleau, 1993] Pomerleau, D. (1993). *Neural Network Perception for Mobile Robot Guidance*. Kluwer Academic.
- [Quinlan, 1993] Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufman, San Mateo.
- [Reeves, 1999] Reeves, T. (1999). A Research Agenda for Interactive Learning in the New Millenium. In *Proceedings of the World Conference on Educational Multimedia, Hypermedia & Telecommunications (EdMedia99)*.
- [Resnick, 1997] Resnick, M. (1997). *Turtles, termites, and traffic jams: explorations in massively parallel microworlds*. MIT Press.
- [Rowe et al., 1967] Rowe, J. L., Lloyd, A. C., & Winger, F. E. (1967). *Gregg typing, 191 series*. Gregg Division, McGraw-Hill, New York.
- [Rumelhart et al., 1986] Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, 323.
- [Russell & Norvig, 1995] Russell, S. J. & Norvig, P. (1995). *Artificial intelligence : a modern approach*. Prentice Hall, Englewood Cliffs, N.J.
- [Schank & Cleary, 1995] Schank, R. & Cleary, C. (1995). *Engines for Education*. Lawrence Erlbaum Associates, Hillsdale, NJ.
- [Schank, 1981] Schank, R. C. (1981). Failure-driven memory. *Cognition and Brain Theory*, 4(1).
- [Shiri-A. et al., 1998] Shiri-A., M. E., Aïmeur, E., & Frasson, C. (1998). Case-Based Student Modelling: unaccessible solution mode. In *Conference internationale sur les nouvelles technologies de la communication et de la formation (NTICF'98)*.
- [Sklar, 2000] Sklar, E. (2000). The Design of the CEL System. Department of Computer Science Technical Report, *in progress*, Brandeis University.
- [Sklar et al., 1999] Sklar, E., Blair, A. D., Funes, P., & Pollack, J. B. (1999). Training Intelligent Agents Using Human Internet Data. In *Proceedings of Intelligent Agent Technology (IAT-99)*.

- [Sklar et al., 1998] Sklar, E., D.Blair, A., & Pollack, J. B. (1998). Co-Evolutionary Learning: Machines and Humans Schooling Together. In *Workshop on Current Trends and Applications of Artificial Intelligence in Education: 4th World Congress on Expert Systems*.
- [Sklar & Pollack, 1998] Sklar, E. & Pollack, J. B. (1998). Toward a Community of Evolving Learners. In *Proceedings of the Third International Conference on the Learning Sciences (ICLS-98)*.
- [Sklar & Pollack, 1999] Sklar, E. & Pollack, J. B. (1999). Demonstrating a Community of Evolving Learners. In *Interactive Presentation at Computer Supported Collaborative Learning (CSCL-99)*.
- [Sklar & Pollack, 2000a] Sklar, E. & Pollack, J. B. (2000a). An evolutionary approach to guiding students in an educational game. In *Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior (SAB-2000)*.
- [Sklar & Pollack, 2000b] Sklar, E. & Pollack, J. B. (2000b). A Framework for Enabling an Internet Learning Community. *Journal of International Forum of Educational Technology & Society, Special Issue on On-line Collaborative Learning Environments, to appear*.
- [Slavin, 1992] Slavin, R. E. (1992). When and why does cooperative learning increase achievement? Theoretical and empirical perspectives. In Hertz-Lazarowitz, R. & Miller, N., editors, *Interaction in cooperative groups: The theoretical anatomy of group learning*, pages 145–173. Cambridge University Press.
- [Slavin, 1995] Slavin, R. E. (1995). *Cooperative Learning: Theory, Research, and Practice*. Allyn & Bacon.
- [Snyder, 1994] Snyder, T. (March 1994). Blinded By Science. *The Executive Educator*.
- [Soloway, 1991] Soloway, E. (1991). How the Nintendo Generation Learns. *Communications of the ACM*, 34(9).
- [Soloway et al., 1981] Soloway, E. M., Woolf, B., Rubin, E., & Barth, P. (1981). Meno-II: An intelligent tutoring system for novice programmers. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI)*.
- [Stanchev, 1993] Stanchev, I. (1993). *From decision support systems to computer supported collaborative work*. Elsevier Science Publishers.
- [Stern et al., 1997] Stern, M., Woolf, B., & Kurose, J. F. (1997). Intelligence on the Web? In *Proceedings of the 8th World Conference of the AIED Society (AIED'97)*.

- [Suthers & Jones, 1997] Suthers, D. & Jones, D. (1997). An Architecture for Intelligent Collaborative Educational Systems. In *Proceedings of the 8th World Conference of the AIED Society (AIED'97)*.
- [Turing, 1963] Turing, A. (1963). Computing Machinery and Intelligence. *Computers and Thought*.
- [Typodrome, 1997] Typodrome (10 January 1997).
- [VanLehn, 1983] VanLehn, K. (1983). Human procedural skill acquisition: Theory, model, and psychological validation. In *Proceedings of the National Conference on AI*.
- [VanLehn et al., 1998] VanLehn, K., Niu, Z., Slier, S., & Gertner, A. (1998). Student modeling from conventional test data: A Bayesian approach without priors. In *Proceedings of the 4th Intelligent Tutoring Systems Conference (ITS'98)*, pages 434–443.
- [Wallace, 1990] Wallace, C. S. (1990). Classification by Minimum-Message-Length Inference. In *Proceedings of the International Conference on Computing and Information*.
- [Walters & Hughes, 1994] Walters, J. & Hughes, B. (1994). Camp MariMUSE: Linking Elementary and College Students in Virtual Space. In *Proceedings of the National Educational Computing Conference*.
- [Wyeth, 1998] Wyeth, G. (1998). Training a Vision Guided Robot. *Machine Learning*, 31.