ProgrammingLand: A Visualization Enhanced Hypertextbook

Curt Hill [1], Brian M. Slator [2], and Vijayakumar Shanmugasundaram. [3].

Abstract - **ProgrammingLand is a visualization enhanced hypertextbook designed and used to teach introductory computer science courses. It is based upon a MOO with a web browser interface. It functions as an online resource that may replace both the course textbook as well as a Learning Management System.**

The system may provide supplementary content material, such as a textbook, to an otherwise conventional course. It may also serve as the entire body of content in a distance education course. This content will generally be in the form of text, but may include graphics, Java applets and other types of interactive exercises to teach and reinforce this material.

The system is organized into hierarchical lessons. It monitors each student's progress through an instructor-chosen list of lessons. The instructor may choose assignments for the system to give a student upon completion of the lesson. Like a textbook, different instructors may choose to use different material or different features that are present in the system.

ProgrammingLand has been used at several different institutions for the last few years and is freely available under the GNU license.

Index Terms – Computer science education, educational technologies, hybrid courses, hypertextbooks and online education. .

INTRODUCTION

In a day when technology is rapidly shaping and reshaping society, the need for effective education is difficult to overstate. It is important to marshal the resources available, including technology, to come to the assistance of an educational system that is scarcely changed in the last five centuries. The subject matter that is taught has changed, but the techniques used to teach it are all too traditional. What is described in this paper is one system to utilize that technology to engage and instruct students as well as assisting the instructors supervising their education.

ProgrammingLand [1] is one such system that also fulfills the requirements of a Visualization-based Computer Science Hypertextbook [2] (VizCoSH). The requirements for a VizCoSH are that it be text based, have significant structure, exhibit some features of a textbook other than just text and contain links leading towards the interactive visualizations[2]. ProgrammingLand is based upon MOO technology with a web interface. (MOO is an acronym for MUD Object Oriented and MUD is for Multiple User Domain.) It is organized in terms of content-based lessons. Any Java applet may be embedded within the system giving visualization capability, although other interactive objects may also be used. It also possesses features of a Learning Management System (LMS).

The form of this paper is to give an extended background on the functioning of the system, then to describe how the visualizations were integrated into this system and then some results of the use of this system.

THE SYSTEM

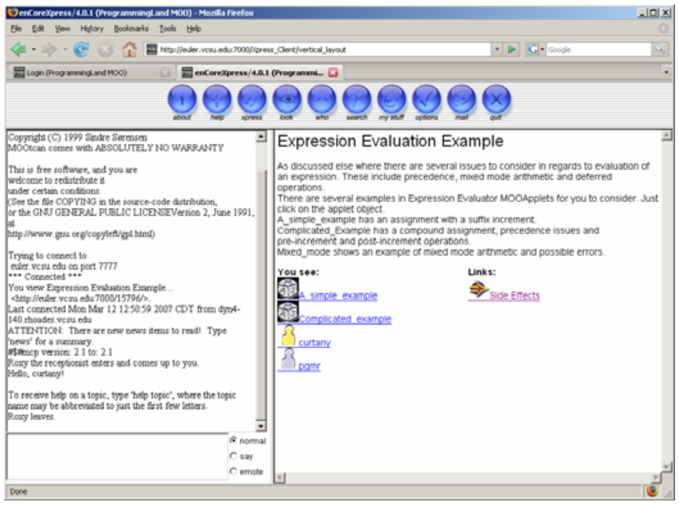
ProgrammingLand requires both a MOO[3] server and a conventional web server. MOO servers have been used for online interactive games as well as online social projects. The MOO provides a room and exit based model and ProgrammingLand adopts a museum metaphor calling the rooms "exhibits." The student sees the content text by browsing through the museum. A MOO is inherently a text-based system using the Telnet protocol, but an interface provided by [4] allows any Java enabled browser to utilize the system. With such an interface, the system fits into the point and click model that most students expect. Figure 1 shows this interface.

The image in Figure 1 shows a web browser showing three panes. The top pane is composed of buttons for common tasks. The left hand pane shows some MOO responses and has a bottom pane for the person to type in commands. The right hand pane is an HTML display of the current room. The top of this is the room description, which is where text content is normally displayed. Below the text are two columns of icons. Those on the left are objects that exist within this room. The first two of these are applets awaiting execution and the second two are persons within the room. On the right is a hyperlink to exit the room.

The person viewing the room will presumably read the content in the right pane. If this person is a student he or she will be given credit for visiting this exhibit. The person may click on either applet, and it will fill the right hand pane. They may use the left hand pane to talk to the other person in the

[1] Curt Hill, Mathematics and Computer Science Department, Valley City State University, Valley City, ND., Curt.Hill@vcsu.edu
[2] Brian M. Slator, Computer Science Department, North Dakota State University, Fargo, ND., Slator@cs.ndsu.edu
[3] Vijayakumar Shanmugasundaram, Computer Science Department, Concordia College, Moorhead, MN., shanmuga@cord.edu

room. They may also click on the link to exit the exhibit or on any of the buttons above.


enCoreXpress/4.0.1 (ProgrammingLand MOO) - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://euler.vcsu.edu:7000/Xpress_Client/vertical_layout
Google
Login (ProgrammingLand MOO)
enCoreXpress/4.0.1 (Programmi...
about
help
xpress
look
who
search
my stuff
options
mail
quit
normal
say
emote
Done

```
Copyright (C) 1999 Sindre Sørensen
MOOtcan comes with ABSOLUTELY NO WARRANTY

This is free software, and you are
welcome to redistribute it
under certain conditions.
(See the file COPYING in the source-code distribution,
or the GNU GENERAL PUBLIC LICENSEVersion 2, June 1991,
at
http://www.gnu.org/copyleft/gpl.html)

Trying to connect to
 euler.vcsu.edu on port 7777
*** Connected ***
You view Expression Evaluation Example...
 <http://euler.vcsu.edu:7000/15796/>.
Last connected Mon Mar 12 12:50:59 2007 CDT from dyn4-
140.rhoades.vcsu.edu
ATTENTION:  There are new news items to read!  Type
'news' for a summary.
#$#mcp version: 2.1 to: 2.1
Roxy the receptionist enters and comes up to you.
Hello, curtany!

To receive help on a topic, type 'help topic', where the topic
name may be abbreviated to just the first few letters.
Roxy leaves.
```

Expression Evaluation Example

As discussed else where there are several issues to consider in regards to evaluation of an expression. These include precedence, mixed mode arithmetic and deferred operations.
There are several examples in Expression Evaluator MOOApplets for you to consider. Just click on the applet object.
A_simple_example has an assignment with a suffix increment.
Complicated_Example has a compound assignment, precedence issues and pre-increment and post-increment operations.
Mixed_mode shows an example of mixed mode arithmetic and possible errors.

You see:
A_simple_example
Complicated_example
curtany
pgmr

Links:
Side Effects

FIGURE 1
SCREEN SHOT OF A STUDENT IN PROGRAMMINGLAND.

Perhaps the most important contribution of the MOO server is the scripting language that it provides. Everything in a MOO must be an object – rooms, exits, students, instructors, applets and many other items. These objects may have both properties and methods, which are programmed in a C-like scripting language. This allows a MOO to be customized in a way that is foreign to most LMS systems. The bulk of this paper describes the customizations of ProgrammingLand.

The MOO server supplies most of the services required; however, a web server on the same machine is required for embedded pictures, the lesson map and applets, the latter two will be discussed later.

I. Lesson Structure

There are several classes of users in ProgrammingLand, the two most important are student and instructor. A student object records what courses the student is currently taking within ProgrammingLand, what exhibits have been visited, what interactive objects have been exercised and similar information. The instructor object may create student objects, create and maintain course objects, enroll students in a course, specify lessons for the course, create and administer tests within a course and as well create new exhibits and lessons.

There are also several classes of rooms in the system. The most common by far is the lecture room. This is the basic exhibit of the system. The content that the student will read in a lecture room allows ProgrammingLand to replace the traditional textbook. The usual practice in this system is for a lecture room to have a short amount of text, typically one to three paragraphs. A lecture room may also have one graphic item in it, such as a picture in any of the common formats. The lecture rooms are linked together by related content, so what may be a fragment of a chapter in a textbook is several lecture rooms. The lecture rooms may be organized into a lesson.

Unlike a textbook, which is mainly a linear system, there are more paths one though the related exhibits. Students may choose their own paths through the material, which helps them to have a feeling of empowerment and engagement. The system records each student's progress through the material by recording on the student object that the visit occurred. This information may be later used to determine if the student has satisfied the requirements of the lesson.

In a classroom situation a lesson may have numerous components. These might include a prerequisite lesson, a motivation for the importance of the topic, a brief overview of the material, a more detailed explanation of the material, examples of the usage of the new material, an exercise to gain experience, and/or a test or assignment to prove mastery. ProgrammingLand lessons may have all of these as well.

A lesson room is a different type of object from the lecture room. Although it will have content text like any exhibit, it has a variety of other properties as well. It is typically the only way to enter the content material in a cluster of lecture rooms. Lesson rooms have properties that describe the contained lecture rooms and requirements of the lesson. When a student completes a lesson, a record of this is placed on the student of object. The requirements of a lesson may include a variety of events that the system monitors.

The most common item in the requirements is the visitation of various lecture rooms. The system cannot determine if the student has read or comprehended the material, but it knows exactly which exhibits have been visited. The lesson may contain more lecture rooms than it actually requires. The instructor will determine which exhibits contain the essential material and which contain material that is optional.

A lesson may require other lessons as well; other lessons may be prerequisite material. Certain types of content may not be easily understood without previous material and ProgrammingLand can enforce that this material has been completed.

Lessons may also be hierarchical. A lesson may be composed of smaller lessons that must also be covered. This is not a foreign view; a course is just a lesson with many sub-lessons, and these sub-lesson may in themselves also have sub-lessons. At the lowest level, a lesson that requires no other sub-lessons should be possible to complete in less than twenty minutes.

Lessons may also require the student to use one of several of interactive objects. Some of these are created purely using the scripting language and others are specialized Java applets. These objects constitute an exercise that a student must finish in order to gain credit for the effort. The Java applets are specialized in the sense that they must be able to recognize the point where the student has accomplished something meaningful and then communicate that event back to the MOO.

A course is yet another object. It lists the students who are enrolled in the course and the lessons that must be completed during the course. These lessons are usually high level lessons with multiple sub-lessons. The instructor places the students into the course with the lessons that will be required.

It is not unusual for two instructors to teach the same course with the same textbook, yet approach the material in quite different ways. ProgrammingLand allows for this variance in behavior in several ways. The selection and order of the course's lessons is the sole decision of the instructor. The choice of lesson requirements is another means of making the course support the instructor's approach. Each lesson room in the system has a standard set of requirements. These are created by the person who created the rooms and organized them into a lesson. A subsequent instructor may use this set of requirements or generate his or her own. These variant requirements are referenced by the course object. Students in that course will only see the requirements accepted by that course.

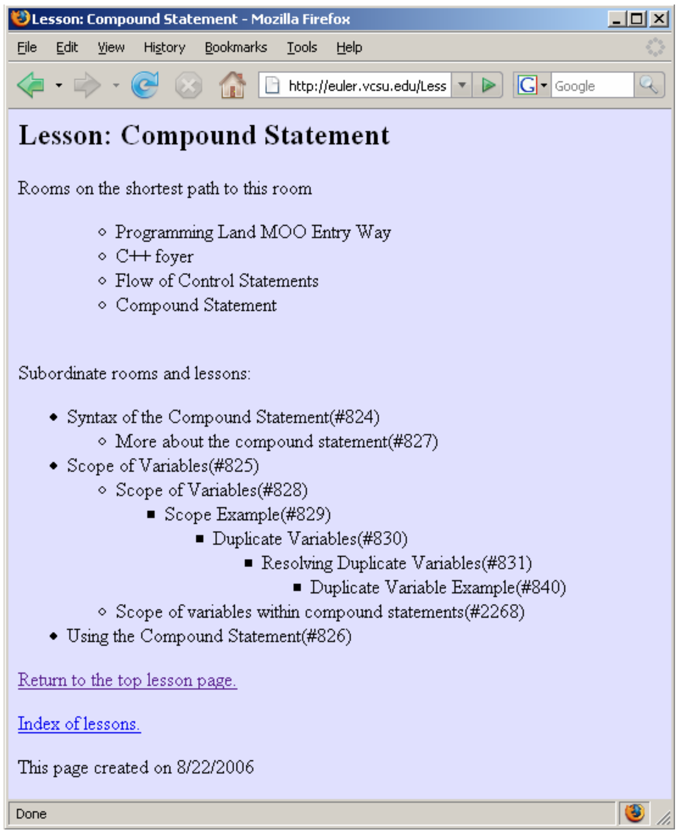
The students are linked to the course as well, so at any time they may inquire as to their course status. One of these requests is a listing of the names of the lessons and which of these have been completed.

ProgrammingLand has several features comparable with an LMS. A course may have an optional grade book as well as give tests and surveys. The grade book records completion of lessons and scores on tests and other assignments. A test may have randomized multiple choice questions that are scored by the system or instructor graded questions. In a conventional class these may be omitted in favor of traditional grade book and tests, or it may be the only form of recording grades. Of course students may only have access to their own scores and grade.

One thing the MOO lacks from the definition of a VizCoSH is a table of contents and index. This ability is supplied by an external lesson map. The lesson map is a series of linked web pages that describes lessons in the MOO. For each lesson the lesson map shows the shortest path to the lesson and the subordinate rooms or lessons. Thus a student may enter the MOO, find the lesson that needs to be worked on, then reference the lesson map to find the most convenient way to that exhibit.

The lesson map starts with a page giving the table of contents for each of the major areas of the system. This links to one URL per lesson. Each of these hyperlink back to the table of contents, to any subordinate or prerequisite lessons as well as an index of all lessons. Figure 2 shows a screen shot of a lesson map page. The lesson map is periodically generated via program [5]. The login screen of ProgrammingLand also has a link to the lesson map.

The screen shot in Figure 2 shows one lesson's page. The top portion shows the shortest path to the lesson. The middle shows the subordinate room. None of these are hyperlinked so the requirements of this lesson only include the visiting of lecture rooms. The bottom has a hyperlink to a table of contents of lessons, based upon the wing of the MOO and a hyperlink to the index of lessons.


Lesson: Compound Statement - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://euler.vcsu.edu/Less
Google
Done

Lesson: Compound Statement

Rooms on the shortest path to this room

- Programming Land MOO Entry Way
- C++ foyer
- Flow of Control Statements
- Compound Statement

Subordinate rooms and lessons:

- Syntax of the Compound Statement(#824)
    - More about the compound statement(#827)
- Scope of Variables(#825)
    - Scope of Variables(#828)
        - Scope Example(#829)
            - Duplicate Variables(#830)
                - Resolving Duplicate Variables(#831)
                    - Duplicate Variable Example(#840)
    - Scope of variables within compound statements(#2268)
- Using the Compound Statement(#826)

Return to the top lesson page.

Index of lessons.

This page created on 8/22/2006

FIGURE 2
A LESSON PAGE FROM THE LESSON MAP .

II. Agents

The system has several software agents that make it more student-friendly as well as carry out other important tasks. There are several agents that help students to find where they need to be and another set of agents concerning lesson completion.

Unlike many MOOs, ProgrammingLand has a topical rather than spatial orientation. One moves towards topics of interest rather than up or left. One of the problems in any system like a MOO is a short horizon. A student in an exhibit may only see adjacent rooms, i.e., those that are one exit away. The convention of this MOO is that exits that take a person to the entry point of the MOO have a different link color than all the rest. Despite this, students often get lost and cannot find what they want. This is particularly the case with newer students.

The lost agent is designed to help those students who appear to be lost or confused. Every time a student enters a lesson room, a message is sent stating that the student arrived. The lost agent examines the student object and finds the course in which the student is enrolled. It then examines the lessons required by the course and compares that with what the student has completed. If the student is not in one of the lessons not yet completed, then the lost agent concludes the student is lost.

A student that is marked as lost then receives a visit from the lost agent. The agent appears as another person in the MOO and comes up to the student and asks if assistance is desired. A student may be either genuinely lost or merely curious about the topics seen, so the ability to decline help is essential. If the student answers in the affirmative, the lost agent will transport the student into the first uncompleted lesson and then leave. If the student declines, the help the agent will leave without changing the student's location.

A student that has declined an invitation for help does not want the agent to be continually appearing and asking the question – that would be annoying. The agent will then ignore the next ten messages that this student has visited a lesson that is not pertinent.

Some students are not lost as far as the lost agent is concerned – they are in the correct lesson, but they are making no progress towards finishing the lesson. The short horizon problem still exists for such students. They may know that they have a room to visit or an object to exercise, but cannot seem to find it. For these students the aimless agent monitors their progress. Aimless looks for the same student visiting the same lesson room with no change in the progress towards finishing the lesson. The lesson must be required for the course and be one of those that is among the first of the unfinished lessons. Aimless must keep a history of each student's progress with a particular lesson. When the lesson is completed, that history record may be deleted.

The aimless agent also receives notification for each student that appears in a lesson room. When the lack of progress status is confirmed, the agent visits the student in a manner similar to the lost agent. Aimless also offers to move the student to a more pertinent room. Like the lost agent, the student's declining of help will prevent a number of future visits.

Students entering any lesson room trigger a number of actions, as has already been seen. Of course, the students are unaware that any such action occurred unless an agent appears. Two others are concerned with the completion of the lesson. A student leaving a lesson room for a room that is not part of the lesson is examined for completion of the lesson. If the lesson is not completed, but the only requirements left are visiting rooms, then the quiz agent will visit the student. If the student visits a lesson and this is the first visit since completing the lesson, then a roving goalie may be summoned.

The quiz agent is an aid to students who may have mastery of the material without having visited the required rooms. The quiz agent will approach the student and ask if the student wants to take a quiz to show mastery. As with other agents the student may answer yes or no. If they accept the offer to take the quiz they are transported to a quiz room and may take a five question multiple choice quiz. If they answer at least four correctly they receive credit for the lesson, even though they have not completed the stated requirements. These quizzes are generated by the system from questions in the exhibits. The quiz room cannot be accessed in any way other than by the agent. It is a room with no entrances.

A roving goalie is an agent that delivers an assignment. It is summoned by a combination of events starting with the student entering the lesson room after having completed one of the requirement sets. The lesson room finds the course in which the student is enrolled and accesses the course object. The course object contains a reference to a dispatcher object, and this object is notified that the particular student has completed a particular lesson. The dispatcher then determines if there is an assignment available for that lesson. If so, the dispatcher sends a roving goalie to visit the student. The goalie will approach the student and give the assignment.

ProgrammingLand currently only has content for the introductory programming classes, so an assignment is usually a programming assignment that is completed outside of the system. However, any assignment that may be described in a web page may also be given. This has included worksheets that the students fill in and return, among other things. Not every lesson is included within the dispatcher, so most lessons are completed without the visit of a roving goalie. The roving goalie may also have the ability to post a score in the course gradebook. In such cases the message given to the student is usually congratulatory in tone.

A course may be customized both for lessons and the assignments given upon completion of a lesson. The dispatcher object is referenced through the course object, so several courses could share the same dispatcher or each course could have a separate dispatcher.

Observe the difference between use of this system and a conventional textbook. With a textbook the instructor usually deduces which students are reading the textbook by their eventual grades. In ProgrammingLand those students who are not doing their reading never get an assignment. A simple check determines if students handed in the assignment they received or one they borrowed from friends.

VISUALIZATIONS

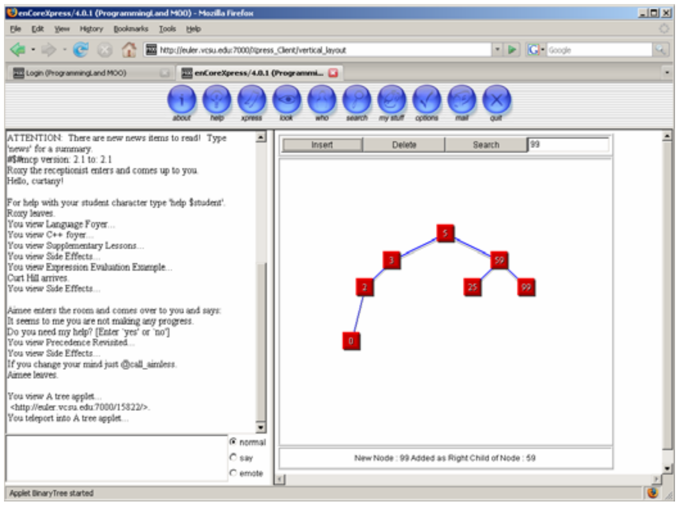
The maxim is that a picture is worth a thousand words, and that an animation is worth a thousand pictures. Although the pedagogical value of this has not been precisely quantified, there has been considerable interest in Java applets that show many important concepts [6]. It is one thing to display a valuable applet on a web page; it is even better to be able to assure oneself that the student has seen it – and better yet to have assurance that the student has executed the applet far enough to see all that is useful. This was the intent when visualizations were added to ProgrammingLand in 2006.

Prior to applets in ProgrammingLand there were a variety of interactive objects, the most common of which was the code machine. Such an object contained a portion of programming language code. It could display this code, it could explain it on a line-by-line basis, or it could simulate an execution of it on a line-by-line basis. Completion of the explanation or simulation could be made a requirement of any lesson.

The creation of the MOOApplet broadened this approach. The problem with an arbitrary applet is that the system cannot determine how well the student has exercised it. The system could only know that they had started it – nothing more. The MOOApplet is a descendent of Applet and provides an additional service. It has a method that will signal the MOO that a particular student has completed what needs to be seen in the applet.

The MOOApplet is actually two pieces, an object in ProgrammingLand and the Java applet, which is external to the MOO. When a student clicks on a MOOApplet the object generates the needed HTML to cause this applet to run in the right hand side pane of the enCore client. The applet itself cannot be stored in the MOO, so the client then fetches the applet itself using the web server that is on the same machine. Of course the student sees none of the behind the scenes manipulation – they click on the applet object and it appears in the pane of the client. Figure 3 shows the execution of a binary tree applet.

In the applet of Figure 3 there are three buttons: insert a value, delete a value and search for a value. Since this paper is not in itself a VizCoSH, the animation of these three operations cannot be seen. In this particular applet, the student should insert several items, delete several items, and search for one item that is found and another that is not. When these actions have occurred, the MOO is notified that the applet has been successfully executed. The student does not receive an indication that this line has been crossed until the display of requirements in a lesson room. The room that contains the object does has instructions to this effect.


enCoreXpress/4.0.1 (ProgrammingLand MOO) - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://euler.vcsu.edu:7000/Xpress_Client/vertical_layout
Login (ProgrammingLand MOO)
enCoreXpress/4.0.1 (Programmi...
about
help
xpress
look
who
search
my stuff
options
mail
quit
Insert
Delete
Search
99
5
3
59
2
25
99
0
normal
say
emote

```
ATTENTION:  There are new news items to read!  Type
'news' for a summary.
#$#mcp version: 2.1 to: 2.1
Roxy the receptionist enters and comes up to you.
Hello, curtany!

For help with your student character type 'help $student'.
Roxy leaves.
You view Language Foyer...
You view C++ foyer...
You view Supplementary Lessons...
You view Side Effects...
You view Expression Evaluation Example...
Curt Hill arrives.
You view Side Effects...

Aimee enters the room and comes over to you and says:
It seems to me you are not making any progress.
Do you need my help? [Enter `yes' or `no']
You view Precedence Revisited...
You view Side Effects...
If you change your mind just @call_aimless.
Aimee leaves.

You view A tree applet...
 <http://euler.vcsu.edu:7000/15822/>.
You teleport into A tree applet...
```

New Node : 99 Added as Right Child of Node : 59

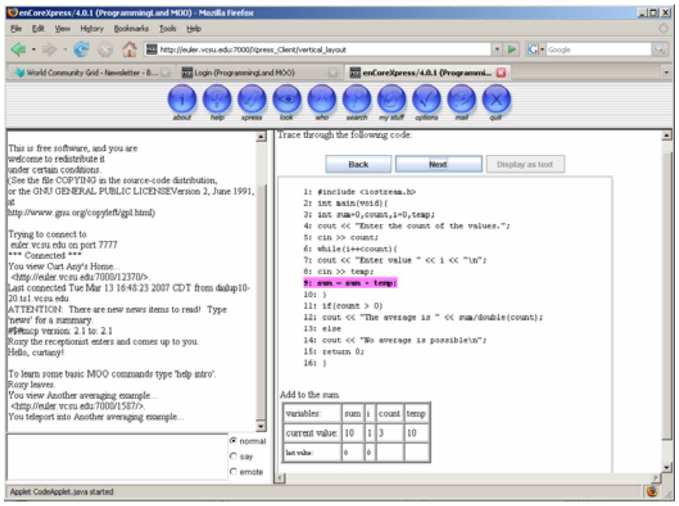
Applet BinaryTree started

FIGURE 3
A BINARY TREE APPLET BEING EXECUTED.

Converting an ordinary applet into a MOOApplet is not difficult, given the source for the original. First, the ancestry of the applet must be changed to a MOOApplet. This is rather painless, since a MOOApplet is a descendent of Applet. Second, the applet must determine what constitutes successful completion. In the binary tree applet, each button click is recorded and the results are tested. As soon as the counts reach the needed values, the MOOApplet method is executed and generates the signal. The applet logic determines how easy or difficult this may be, but two hours of time to convert an applet is about average.

The code machines and some of the other objects that had been purely MOO-based have been converted to MOOApplets as well with a new object name of code applet. Most of the applets available on the web are intended for classes more advanced than the introductory programming class, such as a data structures course. The code machines and new code applets target the introductory student. However, this applet was different than the previous applet. The normal MOOApplet is a usually a conversion of an existing applet and each one is different. The binary tree applet shown in Figure 3 may exist in a C++ data structures exhibit or a Java data structures exhibit, but nowhere else.

The code applet is designed so that one applet may handle any piece of code. All of the needed information comes in through applet parameters generated by the code applet object. Figure 4 shows a code machine applet during a simulation of an execution. There are over 200 code applets in ProgrammingLand.


enCoreXpress/4.0.1 (ProgrammingLand MOO) - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://euler.vcsu.edu:7000/Xpress_Client/vertical_layout
World Community Grid - Newsletter - 8...
Login (ProgrammingLand MOO)
enCoreXpress/4.0.1 (Programmi...
about
help
xpress
look
who
search
my stuff
options
mail
quit
Back
Next
Display as text
normal
say
emote

```
This is free software, and you are
welcome to redistribute it
under certain conditions.
(See the file COPYING in the source-code distribution,
or the GNU GENERAL PUBLIC LICENSEVersion 2, June 1991,
at
http://www.gnu.org/copyleft/gpl.html)

Trying to connect to
 euler.vcsu.edu on port 7777
*** Connected ***
You view Curt Any's Home...
 <http://euler.vcsu.edu:7000/12370/>.
Last connected Tue Mar 13 16:48:23 2007 CDT from dialup10-
20.ts1.vcsu.edu
ATTENTION:  There are new news items to read!  Type
'news' for a summary.
#$#mcp version: 2.1 to: 2.1
Roxy the receptionist enters and comes up to you.
Hello, curtany!

To learn some basic MOO commands type 'help intro'.
Roxy leaves.
You view Another averaging example...
 <http://euler.vcsu.edu:7000/1587/>.
You teleport into Another averaging example...
```

Trace through the following code:

```
#include <iostream.h>
int main(void){
int sum=0,count,i=0,temp;
cout << "Enter the count of the values.";
cin >> count;
while(i++<count){
cout << "Enter value " << i << "\n";
cin >> temp;
sum = sum + temp;
}
if(count > 0)
cout << "The average is " << sum/double(count);
else
cout << "No average is possible\n";
return 0;
}
```

Add to the sum.

variables:	sum	i	count	temp
current value:	10	1	3	10
last value:	0	0		

Applet CodeApplet.java started

FIGURE 4
A CODE APPLET IN A SIMULATED EXECUTION.

CONCLUSIONS

There have been several studies on the effectiveness of ProgrammingLand, although more will be helpful. Reference [7] compared courses at four institutions and found that a course using ProgrammingLand did not suffer, despite having no textbook. Reference [1] compared two sections, taught by the same instructor, one using a textbook and the other ProgrammingLand. The results of this study were also positive. Agent effectiveness was considered in [8]. This study indicated the not surprising fact that stronger students were moved by agents less than weaker students. Moreover, the acceptance of the invitation to be transported did not decline over time, indicating that the students found the help meaningful.

There is limited data covering the first two semesters when visualizations were used to enhance the system, due to small class sizes at the institution where the full lesson structure is used. This data is summarized in Table I.

TABLE I
CORRELATIONS BETWEEN COURSE AND MOO DATA

MOO Event	Course Scores	Tests Scores	Program Scores
Connections	0.446	0.153	0.548
Exhibits	0.919	0.748	0.961
Assignments	0.958	0.816	0.985
Events	0.918	0.740	0.961
Traces	0.627	0.565	0.632

Table I shows the correlations between events that MOO collects and various data from the course itself. Although the MOO may maintain a grade book and collect scores on the students accomplishments, this was not done in this study. The three pieces of data that are extracted from the course data is the total score of the students, the test scores and the graded program scores. This is correlated with the number of times the student has connected with ProgrammingLand, the number of exhibits visited (a revisited exhibit is counted a second time), the number of assignments, the number of events and the number of code machine traces that were completed. The number of assignments given is a measure of high level lessons that make an assignment upon completion. Events include completion of any lesson or the completion of any interactive objects, regardless of whether it is required by the course.

These numbers are similar to previously done correlations. The poor correlation of connections with any objective course score may at first glance seem surprising. However, a student who connects does not necessarily accomplish any useful learning. The rooms visited count is better, but not as useful as desired. A student may run up their exhibit count by circling through rooms already seen and not assimilate any new knowledge. The assignments given shows completed high-level lessons. This measure shows direction and diligence in using the system. The events count is like the exhibit count in that it may be artificially increased by completing things that are irrelevant to the course. The trace count shows the number of graphical code machine traces.

The high correlation between all of these measures, except connections, suggest that the system is an effective teaching environment. Unfortunately, the limited number of students prevents a stronger statement, nor can we say much about how each component contributes to the whole.

Anecdotal evidence also exists. Students seem to like the system partially because it is more engaging, and partially because they save the expenses of a textbook. There is also some evidence of an enrollment increase because of its use.

There are some disadvantages as well. Every college student knows how to use a textbook, but something must be learned in order to use ProgrammingLand. Some class time is required to familiarize students in a conventional class, while online students need an orientation session.

ProgrammingLand has shown itself to be a robust and user-friendly means to dispensing content information for a course. It possesses the characteristics of a VizCoSH, engages students, while transparently monitoring their use of the system. It has been positively received by students and received endorsements from faculty. The system is also freely available by contacting the first author.

ACKNOWLEDGMENT

We extend our thanks to those people who have graciously put their Java visualization applets and source code on the web. The binary tree applet was originally written by John Kloss [9]. Our thanks, in advance, for others whose applets will prove useful in the future.

ProgrammingLand has been supported by National Science Foundation Grant EIA-0313154 and ND-EPSCoR through the FLARE program under EPS-9874802. The authors gratefully thank and acknowledge both of these organizations.

REFERENCES

[1] Hill, C., Slator, B. M., Shanmugasundaram, V., “Measuring the Effectiveness of ProgrammingLand.” Proceedings of IASTED Conference on Web Based Education (WBE 2007). Chamonix, France, March 2007.

[2] Rößling, G., Naps, T., Hall, M. S., Karvirta, V., Kerren, A., et. al., "Merging Interactive Visualizations with Hypertextbooks and Course Managagment", *Inroads - SIGCSE Bulletin,* Vol 38, No 4., December 2006, pp.166-181.

[3] Curtis, P., “Mudding: Social Phenomena in Text-Based Virtual Realities.” *Proceedings of the Conference on Directions and Implications of Advanced Computing* (sponsored by Computer Professionals for Social Responsibility), 1992.

[4] Holmevik, J. R., Haynes, C., “enCore, Open Source MOO project”. http://lingua.utdallas.edu/encore/index.html Date accessed 9 January 2006.

[5] Hill, C., Shanmugasundaram, V., Miteva, M.., “Database Tools to Administer Programming Land.” *ISCA 18th International Conference on Computer Applications in Industry and Engineering (CAINE 2005)* November 2005, Honolulu, Hawaii, pp. 410-413. [CD-ROM]

[6] Boroni, Christopher M., Frances W. Goosey, Michael T. Grinder, and Rockford J. Ross. Engaging Students with Active Learning Resources: Hypertextbooks for the Web. In *Proceedings of the 32nd ACM SIGCSE Technical Symposium on Computer Science Education* (SIGCSE 2001), Charlotte, North Carolina (2001), ACM Press, New York, pp. 65-69.

[7] Hill, C. D., Slator, B. M., Daniels, L. M., “Using and Validating ProgrammingLand” *Proceedings of the 7th IASTED International Conference on Computers and Advanced Technology in Education (CATE-04)*. August 16-18, Kauai, HI, pp. 291-296.

[8] Hill, C., Shanmugasundaram, V., Miteva, M.., “Agents Help Students in ProgrammingLand.” Proceedings of the 11th SIGCSE Annual Conference on Innovation and Technology in Computer Science Education. June 26-28, 2006, pp. 183-187.

[9] Kloss, J,. Home Page. http://www.cs.jhu.edu/~jkloss/ Date accessed 13 August 2007.