Evolution of ProgrammingLand

by

Curt Hill

Valley City State University

Curt_Hill@mail.vcsu.nodak.edu

Abstract:

The ProgrammingLand MOOseum is a text-based, virtual environment devoted to the education of Computer Science students in their initial programming classes. It is an online, learner-centered museum of programming that promotes active learning. It serves a purpose similar to certain types of course management software, but is fully customizable in a way that most such software is not. The system provides content material, hosts interactive exercises, monitors the progress of students and gives assignments. It was designed for both distance education and the enhancement of classroom situations.

Introduction.

An ongoing experiment in using virtual worlds for Computer Science Education is described. The project was originally envisioned as a vehicle for teaching introductory programming concepts in a distance education mode. However, it has been extended to usage in a classroom context with good results. The organizational and educational benefits of this approach are described.

The Basic MOO.

ProgrammingLand [Hill and Slator 1998] is built on a MOO, which is a specific form of a MUD or Multiple User Dungeon. MUDs are most commonly used for role playing games of either a fantasy or social nature[Curtis 1992]. In a typical situation a student uses client software to contact the MOO server over the Internet. Each MOO consists of at least two pieces on the server computer. The first is the MOO server itself, which in the case of ProgrammingLand is called WinMOO [Unkel 1997] and runs on the Microsoft Windows 95/98/NT platforms. There are several other such pieces of free software, which are able to host a MOO on other platforms. The second piece is called the MOO core, which is the database that determines the characteristics of the virtual environment. This database is in text form and is read by the server when it starts. The same text database may be used on different platforms provided the password encryption routines are similar.

ProgrammingLand started with a core developed by Highwired Encore [Haynes 1997]. However, a core determines the basic objects of the MOO but does not define very much of the environment. Thus ProgrammingLand has been and continues to be built up to accomplish its goals. A MUD or MOO is usually a text-based virtual world although several projects such as the Geology Explorer[Saini-Eidukat, Schwert and Slator 1999] and the Virtual Cell[White, McClean and Slator 1999] have combined a MOO with other software to provide a graphic interface.

The basic objects of a MOO are rooms and exits. A room represents a location and an exit the means of moving from one location to another. When a person is using a MOO, they use the name or alias of the exit to move from their current room to another. When they arrive at a room the description of that room is displayed to them as well as its visible contents. These contents may include other players or other types of active or passive objects.

Most MOOs and MUDs use a physical metaphor, so exits carry the names of physical directions such as North, Southeast, Up and Down. ProgrammingLand has a conceptual metaphor, which is that of a museum of programming. Therefore the exits are labeled topically rather than directionally. ProgrammingLand has two styles of rooms, which also called exhibits: menu exhibits and normal exhibits. Menu rooms have little real content, but only show the direction to adjacent exhibits of interest and typically have exit names such as a, b, c, etc. Normal exhibits usually only have two or three exits which are usually named with a key word that indicates their contents.

A brief introduction to computer hardware
This exhibit briefly introduces the hardware components that are used. More detailed descriptions are found elsewhere.
Computer hardware such as found in most desktop and laptop computers consists of several distinct pieces that are worth their own exhibits:
a) memory, short term storage
b) the CPU, the center of all computers
c) secondary storage, long term storage
d) devices to communicate with people
e) devices to communicate with other machines
or
x) Return to the introductory topics exhibit

Figure 1: A menu room

In figure 1, the first line is the room's object name. The rest of the lines are the description of the room. If the student were to type *b* then they would arrive in the CPU room, which is normal exhibit and it is shown in figure 2.

CPU
CPU is an acronym for Central Processing Unit. It is indeed the logical center of any computer and usually defines the characteristics of a particular piece of hardware. The CPU is composed of components such as registers, adders, comparators, multiplexors and these are composed of lower level things called logic gates.
The CPU's main task is to execute instructions. An instruction is a command to do some small amount of work, such as adding two numbers, comparing two numbers, moving something from one location in memory to another or to continue execution somewhere else.
These instructions are stored in memory, just like any other data. An instruction specifies what the task is to do and also what operands to work on. All the CPU does is execute an instruction from memory and then execute the next one. This process of executing an instruction is called the fetch execute cycle.
This sounds rather unimpressive and it is. The only real noteworthy thing about the whole process is that it is amazingly fast. A modern computer can typically execute millions or billions of instructions in a single second.
Obvious exits: [exit] to A brief introduction to computer hardware, [back] to Memory, [more] to The Fetch Execute Cycle

Figure 2: A normal room

Also in figure 2, the first line is the name of the exhibit. The last two lines announce the exits that are available. If the student types *exit* they return to the previous exhibit, while *back* and *more* take them to other exhibits.

Players are also objects in the MOO, which has several advantages. The server requires a login with password, so that it can connect a student with their ProgrammingLand object. The student object stores various pieces of information about the student, including the rooms they have visited and several kinds of events that have occurred. This allows substantial record keeping on students, comparable to several kinds of course management software, and is at least as flexible as most of the common ones. When two students are in the same room, each is told of the presence of others. They may also carry on a conversation, for every room of a MOO is a chat room.

There are three levels of character objects. The least privileged are the student characters, whom cannot create or modify objects or examine the properties or verbs of objects. These objects can usually only own themselves. The next higher level is the programmer. Programmers may create new objects and modify or examine any object that they own. Most of the MOO is actually owned by a single programmer. The highest level is that of a wizard. A wizard may examine, modify, delete or create any object regardless of ownership.

The Virtual Lecture.

The MOO that has been described to this point mostly resembles an online textbook although somewhat more active and learner centered than a conventional textbook. One of the metaphors of the MOO is that of a virtual lecture or virtual lesson. Consider the classroom delivery of a lesson to a series of classes. A good lecture will have several components. These may include: motivation as to why this topic is important; an explanation of the new material; examples of the use of the new material; and exercises to reinforce the material. In any given lesson the balance between these and other components will depend on the instructor's perceptions of the students as well as student questions. Unfortunately in a classroom situation some type of typical student must be in sight, which results in uneven satisfaction. Some marginal students may fall behind, some good students may not be challenged, interesting questions cannot always be dealt with because of time constraints. In ProgrammingLand the goal is to provide all of these components, and to some degree, let the student choose which ones they need to experience and in what order. Thus a student may pick and choose how much they want to experience. Of course, there must be some requirements for academic integrity, but this just means the MOO should provide much more information than is required.

A characteristic of a MOO is that everything is an object, with behaviors and attributes (also called properties). The behaviors are the object methods and are also called verbs. The scripting language of the verbs is similar to C and allows almost any characteristic of the MOO to be changed dynamically. Most of this paper describes the results of programming the MOO, although the programming itself is not much discussed.

Interactive exercises.

There are several types of interactive virtual machines that populate ProgrammingLand. These are designed to engage the student in a way that static text is unable. The most common is the code machine. Each code machine contains a segment of programming language code. On command it will display the code itself, a line by line explanation of the code, or the trace of the execution of the code on a line by line basis. This is not a simulation of arbitrary code but a single execution of the code that has been documented. There are several others that are less common but still useful including the workbench, the code repository, the ring toss game, the history jukebox, the code scrambler, the recursive leprechaun, tutor robots and demonstration machines.

Unfortunately, the design and development of new, useful machines is time consuming and requires substantial creativity. Yet the reward is an environment that is both engaging and educationally useful.

The Lesson Structure.

In most cases students are a very peculiar form of consumer. They are paying for a service, yet they are delighted when the producer does not give them the service they paid for. In those years when ProgrammingLand was an option for a programming class and not required they used it rather infrequently. The problem is then how to ensure that the students are actually taking advantage of the resource. A seemingly unrelated problem is determining which are the essential exhibits that a student should have mastered and which are those that are not required for mastery. The common solution to these two problems is the structure of ProgrammingLand lessons.

A lesson in a class presents some content that the students need to master. In the MOO this content is usually spread over a series of exhibits. A lesson should contain all those pieces that were mentioned earlier: motivation, information, examples and exercises. So a typical lesson in ProgrammingLand should contain a few exhibits on why this topic is important, more on the content material, several on examples and some type of device that makes the student rehearse and exercise the knowledge. Typically a single room is the entrance to this set of related exhibits and often that exhibit is a menu room.

There are three separate objects that manage a lesson: the lesson room, the quiz room and the lesson exit. The central item is the lesson room, which is a specialized form of the room object. It records the requirements of the lesson and the location of the quiz room. The requirements are those things that the student needs to complete in order to satisfy the lesson. The events that may be required include a visit to an exhibit, the successful use of a code machine or other interactive object, and the completion of another lesson. Lessons may depend on other lessons, so the required lesson could be a subordinate that is contained by the current lesson or it could be a prerequisite lesson. It should be noted that normally a student cannot distinguish between a regular exhibit and a lesson room. There are a number of properties stored on the lesson room object, such as the requirements of the lesson and the exhibits contained in the lesson, however this extra information is hidden from the view of the student. It is the one place where the information is stored so that the quiz room and the lesson exits may access it there. Only when they use a lesson exit to leave the whole lesson does this come into play.

The requirements of a lesson are those objectives that the instructor desires a student to accomplish in order to gain mastery of the content material. In general, there are several different ways that a student should be able fulfill the instructor’s wishes. Thus the lesson should be able to announce mastery of lesson by the student based on several different sets of actions by the student. The requirements property of the lesson room contains these different sets of actions and is organized as a set of alternatives. Each alternative is a list of any number of the following items: rooms to visit, lessons to complete, or machines to exercise. Satisfaction of one of these alternatives requires that the student complete every item in the group. A student may satisfy any of the alternatives to satisfy the lesson, but to satisfy the alternative they must satisfy everything.

It is not enough to impose these requirements on the students, but it is also necessary to allow them to find out the requirements of each lesson. Therefore the @requirements command is provided that tells them what is necessary for this lesson.

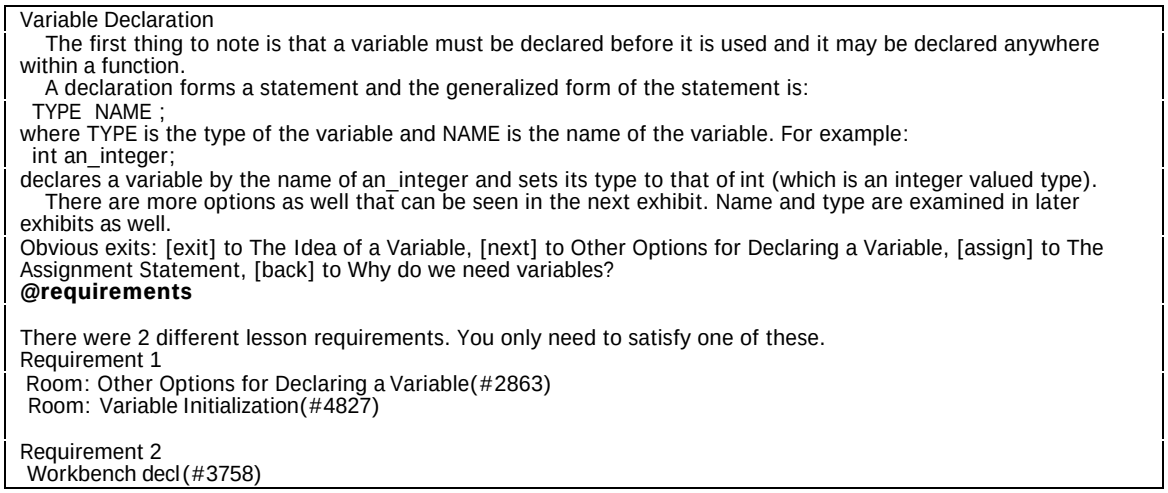

Figure 3: A lesson room and @requirements command

The first part of Figure 3 is the room description, followed by the user @requirements command and the response. The student may either view the two rooms or work through the use of a workbench. A workbench allows them to build statements and then parse them for correctness. If the student uses the @requirements command in any other kind of room, they are told that the exhibit is not a lesson room and thus it has no requirements.

A lesson room usually controls access to every room in the lesson. There is no absolute reason to do so, but it is characteristically the case. It is usually the case that the only exit from inside the lesson to outside the lesson is an exit from the lesson room. There is a specialized exit called the lesson exit which is the focal point for a student to be checked by an agent. This agent checks the student's progress when they leave the lesson. The rooms that they have visited, the machines they have worked and the lessons they have completed are all recorded on the student history. The agent, initiated by the lesson exit, compares what they have done with the requirements stored in the lesson room. If they have satisfied the requirements the credit for the lesson is posted to their history. If they have not done so they are told of at most two deficiencies of the requirements. Since there may be multiple alternatives in the requirements, the agent only uses the first alternative to display deficiencies. If that alternative only contains rooms that should be visited they are given a choice: leave the lesson without credit or take a quiz to show mastery of the lesson material. If they choose to leave, they may come back at any later time and complete the material they are lacking. They never need to redo what they have previously completed, since this information is captured in their history. If the first alternative has requirements like lesson completion or machine use, which are not amenable to satisfaction by quiz, they are not given the choice, only notification that they have not finished the lesson. If the first requirement alternative contains only rooms to visit and the student elects to take the quiz, they are taken to a quiz room referenced by the lesson room. This destination is used instead of the normal exit destination.

Lesson exits may be used in any room within the lesson that is connected to any room outside of the lesson. Thus there may be more than one lesson exit for a lesson, although this occurs infrequently. Furthermore a lesson exit always scans a student, but if the student has previously satisfied this lesson they pass through without notification that they had already passed the lesson.

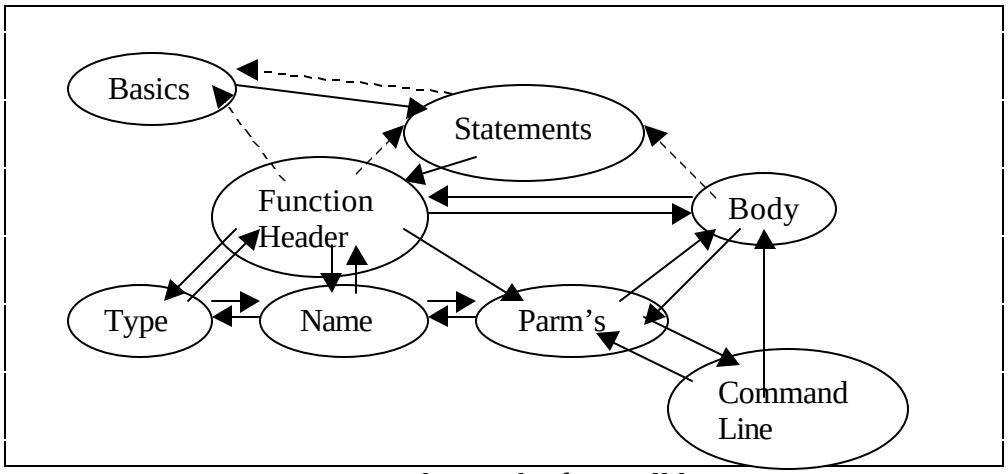

Figure 4: The graph of a small lesson

Figure 4 shows the partial graph of three lessons. A room is represented by an oval and exits by arrows. The largest lesson is called Basics and is a introductory lesson on C++ programs. It contains another lesson, identified here by Statements. Most of the rooms and exits of these two lessons are not shown. Within the Statements lesson is a small lesson on the main function header of a C++ program. This would be encountered well before the students are ready to define their own functions. There are four lesson exits shown, the two from Function Header and the one from Body belong to Function Header, and the one from Statements belongs to Statements. If a student were to take any of the three Function Header lesson exits, then their history would be compared with the requirements of Function Header. Figure 5 shows a student leaving the Body room after having satisfied the requirements for the Function Header lesson.

```
The function body
    Every function needs a body, which is that group of statements that will be executed when the function is
called. A function body is just a compound statement.
    A compound statement starts with an opening brace "{" contains one or more statements and is finished by a
closing brace "}".
    Learning what these statements can be is the largest part of learning C++. The two that the example used were
a cout statement and a return statement which are discussed in the exhibit on statements.
     cout is used to display things on the output screen.
   return ends the function and shows what value the function should return.
   The statements are executed sequentially within the compound statement.
Obvious exits: [exit] to Program and function headers, [statement] to Statements
statement

You have completed the requirements of this lesson, updating your events.
Statements
    An executable statement is a command to perform some action, a non-executable statement usually declares
something for later use. Although many kinds of statements will be discussed in later exhibits, just a few are
discussed here.
    C++ is an expression language. What this means is that any expression can be a statement. (Further
significance of this will be discussed with the assignment statement.) Statements are terminated by semicolons,
that is each statement is concluded by a semicolon.
    Two important statements are discussed in the following exhibits:
a) Output can be accomplished using the predefined variable cout.
b) The return statement ends a function or program.
 or
n) Move on to the Function header exhibit
r) Return to the comment exhibit
x) Exit to the basic exhibit
```

Figure 5: A student leaving a lesson

Figure 5 contains two room descriptions, the Body exhibit and the Statement exhibit. Following the Body exhibit display is the command that the student typed in, in this case statement. Following the blank line is the notification that the student had satisfied the

lesson requirements. If they should come into this lesson again and then leave they will not receive this notification again. However, the @progress command will show them all the lessons they have completed as well as their current goal.

The quiz option is particularly useful for students with more background than average. They can show that they satisfy the material without necessarily visiting every exhibit. The only purpose of a quiz room is to deliver a multiple-choice quiz to the student and the only way into one is by having been sent there by a lesson exit. Thus they are rooms with no conventional entrances. If the student passes the quiz, that is they answered at least four of the five questions correctly, they are given credit for the lesson. A student may only take a quiz offered from a particular quiz room twice, if they fail both then they must satisfy the requirements normally.

The quiz room, like the agent that sends them to the quiz room, accesses the requirements property in the lesson room. It generates the quiz by accessing the first alternative on the requirements property. It checks the rooms that are on this alternative and finds all the rooms that the student has not visited, yet are considered to be essential. Each room in a lesson that is on the requirements list should have as a property one or more quiz questions that pertain only to the content of the room. The quiz room gathers all the questions from the unvisited rooms and randomly reduces them to just five. If there are not five then the lesson room itself has a supply of more general questions that will bring the total up to five. Each question has at least three pieces: the question setup, a list of right answers and a list of wrong answers. The quiz room randomly chooses one of the right answers and four of the wrong answers and then scrambles the order of the answers. It then presents the question and waits for the answer. If the student answers incorrectly the correct answer is shown.

A command has been implemented on the wizard that shows all the events, such as lesson completions, that have been posted to a student. In this way the instructor may monitor the progress of a student or a class of students.

Online assignments.

There is one more aspect of the lesson structure that has not yet been mentioned. As has been discussed, any lesson completion for a student posts the event onto the student character. Another object also receives notification of the lesson completion. This object is called the dispatcher and it takes the lesson number and looks it up in a table of lessons which have offline assignments. If the lesson is not present in this table then the lesson has no offline assignment and the notification to dispatcher may be discarded without action. If the lesson is present in this table, then an agent called a roving goalie is activated after a three second delay. The purpose of the roving goalie is to visit the student and give them a new goal. The interaction is as if another player object met the student, tells them something and then leaves. In practice the roving goalie is not a player object but an agent.

The MOO is a versatile environment for education but it is not a universal environment. In particular, the implementation of a development environment for an arbitrary programming language on a MOO is a daunting task. Development effort and server performance are the two largest arguments against such an effort. Therefore the “out of MOO” assignment has been considered essential for proper programming education. This type of assignment is usually similar to the traditional programming assignment homework, but can be other things as well. For example, in a lesson on

computer hardware, the use of a simulator that demonstrates low level machine operations is an assignment. In any out of MOO assignment the student is to email the results to the instructor.

```
You have completed the requirements of this lesson, updating your events.
C++ foyer
    This is the beginning of a series of lessons on the programming language C++. The following menu of lessons
gives you a choice on what to examine next.
a) Some background in using the MOO in the study of C++ as well as some history of C++
b) C++ basics, such as the form of programs
c) The notion of a variable
d) Some miscellaneous small lessons that follow variables
e) Execution flow of control
f) Function definition and usage
g) Libraries and include statements
h) Advanced topics
  or
m) Go to the C++ Group Work Room
x) Exit to the language foyer

Fred enters the room and comes over to you and says:
You have now completed the form of C++ programs goal.  You now have two tasks ahead of you. The first is the
creation of a program using your compiler. It should be similar to the simple programs that you have seen so far.
You should write a program that displays your name, the name of the state you were born in and full email
address.
Each of the items should be on a separate line and use a delay call at the end to keep the console window on the
screen.
The program should be mailed to your instructor when you are complete. The only thing to mail is the C++
program, which will have an extension of .CPP which should be an attachment on your mail.
After this you should move on to the variables section of the MOO and master that assignment.
Fred leaves.
```

Figure 6: A student receiving an offline assignment

Figure 6 shows what is displayed when a student leaves the Basics (which was part of the graph of Figure 4) after they have completed all the requirements of the lesson. The first line is the indication of their completion and the next fifteen lines is the normal display of the C++ foyer. Approximately three to five seconds after the student has arrived in the exhibit, the next lines appear. The roving goalie named Fred comes up to the student, gives the student the message and then leaves. Fred acts somewhat like a MOO player character but is just a software agent.

The roving goalie does not have just one assignment for all the students who have completed the lesson, but a set of equivalent assignments. The goalie sequentially gives them to successive students. If the goalie has more assignments than students in the class then each will receive a different assignment. This feature was implemented to reduce the undesirable type of collaboration, ie. cheating. Since each student has a different but equivalent assignment the only notification of the assignment is from the roving goalie. However, the student has a @showgoal command which will display the current goal, which is description of the assignment. They also have a @progress command which shows the current and all previous goals.

Administering ProgrammingLand.

The original MOO core had about 100 objects, while the current database exceeds 8000. Ensuring the consistency of such a project can be a problem. A number of approaches have been used. Since students have a different point of view than instructors they are offered points for finding errors in the MOO. It does not seem to be the case that they actively look for problems, but when they find a problem they are much more likely to report it. The MOO itself can be used to find some problems. There are verbs that determine if the quiz questions of a room are properly formatted and the requirements of a lesson are consistent among other things. However, many items are time consuming to

look at using MOO verbs. Each situation requires the creation and debugging of its own verb. A successful solution to this general problem was the MOOMiner project[Hill 1999]. MOOMiner is a Java application that systematically looks at each object and builds a database of its findings. The program logs on as a wizard and makes an ODBC connection to a database. The program then examines each object and records its findings in tables in the database. Since all the communications are through TCP/IP connections the MOO server, the database server and the MOOMiner program may all run on separate computers. Once the database is complete then a number of SQL queries are used to find inconsistencies and other information. Some of the queries include invalid exits (source or destination is not a room), rooms with no exits, objects not owned by the author, etc.

The Lesson Map.

One of the problems of a MOO (or a series of web pages) is the horizon effect. The student can see the room (or page) they are at and they may also see the exits (or links), which provide a preview of those areas that are one move away. However, that is all they can see, the horizon is too close and they have difficulty finding their way to a location beyond the horizon. ProgrammingLand has the convention that every room should have an exit with the name of *exit* and aliases of *x* and *out*, which will take the player at least one step closer to the start of the MOO. This start room is the room where all students begin their explorations, the entrance to the museum. A student may make their home (the place they start each session) in any room, but the default is the entrance to the ProgrammingLand. These exit conventions are somewhat similar to the back button of a browser, except that these exits are fixed paths toward the entryway not a retracing of the history of the student. This is helpful in the sense that it make it easy to get to the starting point of the MOO, but of no help if a student is trying to find an exhibit they have never visited. Their dilemma is that they know what the room name they are looking for by using the @requirements command, they just do not know where to find it. The solution that has been devised is the lesson map.

The lesson map is a series of web pages that contains information about the relationship of the various lesson rooms in the MOO. The lesson map does not contain the content material, just some of the shape of the rooms and exits. The lesson map is viewed with a web browser and not with the normal MOO client software, so no recording of information about what the students have examined can be captured. Instead, there is one page for each lesson room. The page contains information on how to get to the lesson as well as the rooms contained within the lesson. The shortest path to the lesson is shown in terms of the rooms on the path to the lesson. The rooms that are shown in an outline form, with adjacent rooms at the highest level and other rooms indented based on the number of moves it takes to get to them. Any lessons that are accessible from a lesson or contained in a lesson are hot linked. The top-level page is the lessons that are available from the various wings of the MOO and there is also an alphabetical index. Both of these are linked from every page.

The lesson map is generated to make it easy to modify the map when the MOO changes, which is a frequent occurrence. The process works using the following steps. The SQL database is used to obtain a file containing all the rooms and another with all the exits. A C++ program reads these in and builds a memory representation of the

resulting directed graph. In this process it collects all the lesson rooms and generates a page for each as well as the top level and index page.

Enhancement of classroom activities.

ProgrammingLand was designed for application in distance education situations, but has mainly been used as a supplementary resource for residential classes. The student usage of the MOO was mostly outside of scheduled classroom time. Starting in the fall semester of 2000 it has been used during class times as a means of promoting and regulating student collaboration. The objects involved in this are the Group Workroom and several objects known as Take-Turn games. The Group Workroom allows the students to subdivide into small groups for a specific activity, usually working on a Take-Turn game. A Take-Turn object forces students to make moves in a cyclic fashion such as in a board or card game.

A common classroom activity in programming is a tracing exercise. In this activity two or three students execute a piece of code by hand and determine the final value of several variables in a simulated run of the code. This can be effective because it gets students to verbalize what is occurring in the code. This verbalization helps them to understand it better and may expose flaws in their thinking. One problem with such exercises is that several student combinations thwart the desired educational outcomes. A strong student, either in academics or in personality, might dominate a weaker one. In such a situation the exercise reflects just one student and neither student learns effectively. If the exercise gets off track because of early student mistakes, then all the later reasoning is based on faulty earlier information. A collaborative approach in the MOO has some promise for improving this situation and the education outcomes.

The Group Workroom is a special type of room where students may form into smaller groups. Any student may invite any others to become a small group. If the others accept the invitation the group is formed. Joining the group also starts a Take-Turn game, which is the exercise for the students to complete. The Group Workroom gives the same Take-Turn game to each group of students. Currently there are two such Take-Turn games: a code scrambler and a tracing exercise. In the code scrambler, the students are presented with some lines of code that have been randomly scrambled and they have to arrange lines of code in way that will make it do a specified task. These are somewhat limited and are mostly driven by syntax. The tracing Take-Turn game is somewhat more interesting and is also more complicated.

The tracing exercise contains a short program segment and the students are to show the flow of execution through the segment. Each player must perform two tasks in each of their turns. First they must specify which line is executed next. If they guess wrong they are told of their error and their turn continues until they get the line right. After the line to be executed has been established, they must specify what variables are changed, if any, and the new values that these variables take. Any mistakes in choosing a variable that is changed or the new value it will assume ends their turn. The students alternate turns until the trace is complete. At any time any student may display the code segment or the values of all variables before the current statement. They may also use the chat feature to discuss among themselves what is happening and why. The system guarantees that each student takes their turn in order. It announces when a student's turn begins and the effects of other student's choices. Since the MOO maintains the correct values of the variables, the students can query these values even if they made mistakes earlier. Thus

only the effects of a single statement are being considered. When the exercise is over, a record of the number of mistakes made by the individual student is posted on their object.

The Laboratory for Advanced students.

The main use of the MOO is that of a resource for students in the first or second semester of an introductory programming course. The student comments on this part of the experience have been good. The use of the MOO for more advanced courses have had some unforeseen benefits. The MOO has been used for several sections of two Junior level courses. These students have usually had two player objects. One was a conventional student character and the other a programmer character. The former was used to familiarize them with the use of the MOO and also to test the items that their programmer character had created. This usage has had at least two benefits, enlarged horizons and new ideas.

First the MOO is unlike most programming environments so it makes the student think about programming language concepts. The MOO scripting language is a type-less object-oriented language. Thus every object is instantiated and derived from an existing instantiated object unlike the object relationshisp used in C++ or Java, which are the usual languages that these students have seen. The object properties are dynamically typed and the language is interpreted, which makes the language similar to LISP or Scheme even though the syntax looks more like C. These two features alone suffice to broaden the student's horizons.

The second advantage is that students are tremendous source of new ideas and approaches. They lack a fixed mental image of what the MOO can or should be, so they tend to look at things in different ways. Several interesting objects have been developed by students, as projects in the advanced courses. These include the ring toss game, the history jukebox, and the recursive leprechaun. Very often the idea alone is all that proves fruitful. A student devised a code scrambler game. None of the original code was used but the idea was effective when it was made into a take turn game.

Future Work.

The ProgrammingLand MOOseum is currently a text-based environment. However, there are several objects, most notably the workbench, which would greatly benefit from a more graphic approach. The text approach is not particularly intuitive and students have generally avoided workbenches. A re-implementation of the server software seems the most promising as well as costly solution to this problem.

The lesson map is external to the MOO, but an internal map of rooms may be desirable as well. The map could be of a whole wing of the MOO, such as the C++ lessons or just the exhibits that the student has already visited. Since the MOO is continually expanding in exhibits it would be better if this map could also be automatically generated.

There is also an endless need for creativity in conceiving and implementing new machines that have educational value. The desirable machine implements a lecturelet [Culwin 2000]. Such a lecturelet should randomly generate an exercise and then guide the student through it. These are needed to maintain the student's interest as well as get the student thinking about the topics at hand.

References.

Culwin, Fintan (2000). Lecturelets - Web based Java enabled lectures. *SIGCSE Bulletin Sept 2000, Vol 23, No 3, pp. 5-8.*

Curtis, Pavel (1992). 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)

Haynes, Cynthia , and Jan Rune Holmevik, eds (1997), *High Wired: On the Design, Use and Theory of Educational MOOs.* University of Michigan.

Hill, Curt (1999). Extracting Data from a MOO. *Proceedings of the Small College Computing Symposium, 1999.*

Hill, Curt, and Brian M. Slator (1998). Virtual Lecture, Virtual Laboratory or Virtual Lesson. *Proceedings of the Small College Computing Symposium, 1998.*

Saini-Eidukat, B., Schwert, D.P., and Slator, B.M., 1999. The Geology Explorer, a Virtual World to Enhance Learning of Geological Problem-Solving. *Proc. XIV Argentine Geological Congress, Salta, Sept. 19-24, 1999, v. 1, p.87-88.*

Unkel, Christopher (1997). WinMOO. http://www-personal.engin.umich.edu/~cunkel

White, Alan R., Phillip E. McClean, and Brian M. Slator, 1999. The Virtual Cell: An Interactive, Virtual Environment for Cell Biology. *World Conference on Educational Media, Hypermedia and Telecommunications (ED-MEDIA 99), June 19-24, Seattle, WA.*