AN ONLINE COMPUTER SCIENCE INSTRUCTIONAL RESOURCE

Curt Hill
Department of Mathematics and Computer Science
Valley City State University
Valley City, ND, USA
Curt.Hill@vcsu.edu

Brian M. Slator
Department of Computer Science
North Dakota State University
Fargo, ND, USA
slator@cs.ndsu.nodak.edu

Vijayakumar Shanmugasundaram
Concordia College
Moorhead, MN, USA
shanmuga@cord.edu

Lisa M. Daniels
Department of Teacher Education
North Dakota State University
Fargo, ND, USA
Lisa.Daniels@ndsu.nodak.edu

ABSTRACT: An online system named ProgrammingLand, with a variety of useful features is described. The system is modeled after a museum where the students browse through exhibits. These exhibits provide the content material of lower level computer science classes. Although not obvious to the student, each exhibit visited is recorded and a lesson structure verifies that each student must satisfy certain requirements to complete a lesson. The entire system is learner centered so that the students choose what to do next. In addition interactive objects provide educational experiences and promote active learning. Some lessons provide an assignment that must be completed outside of the system, usually but not limited to programming assignments. The system contains a number of software agents to make it easy to use. It also contains features of a more traditional Learning Management System.
The system has been successfully used as a substitute for a textbook and on a more limited basis as a vehicle for fully online, distance education.

KEY WORDS: Educational research, blended education, hybrid education.

1. Introduction

ProgrammingLand is an online system to deliver educational experiences in the lower level computer science courses, as well as provide the support a student or instructor would expect from a Learning Management System (LMS). An educational experience may include content delivery, such as would normally come from a textbook or lecture, interactive exercises which might otherwise be homework or in-class exercises and it is able to deliver assignments the student must complete outside the system. In addition it monitors the student's progress and renders aid both to the student and the support staff of the course.

The use of ProgrammingLand varies depending on the instructor's desires. A limited number of students have used it as their main resource in fully online and distance courses. In this context the instructor mainly corrects assignments and gives assistance where needed. The system functions as the textbook, gives assignments and monitors progress.

ProgrammingLand has traditionally functioned as the online resource in more traditional classes. In this mode it replaces the textbook in a traditional classroom experience. These are blended or hybrid courses, where both a classroom and online experience is required. Our experience is that this approach works much better for the typical college student than the purely online course. This should be no surprise, for it appeals to a wider number of learning styles.

This paper is structured into descriptions of the various components that work together to provide these experiences and services. The following sections will discuss the technology used, the lesson structure of ProgrammingLand, some of the types of interactive objects, the use of software agents and the learning management or course management facilities that are provided.

2. Technology in Use

ProgrammingLand is a blend of old and new technology. The root technology is that of a MOO[1], but the students always use a Java-enabled web browser as their client program. A MOO is an object oriented MUD. Thus it is largely structured as rooms and exits leading from one room to another. The motif of ProgrammingLand is as a museum where students view exhibits and interact with objects that have educational value. Thus in many MOOs the exits are labeled by

direction, such as north or up. In ProgrammingLand the labels indicate the content of the exhibit to be entered.

One of the advantages of a MOO is the ability to create new types of objects and customize the behavior of any object. Like any object-oriented system, MOO objects have properties and methods, the latter usually called verbs. It is usually not possible with a LMS or many online systems to modify its behavior in unanticipated ways. The ability to script objects in new or different ways is the main feature that allows ProgrammingLand to have a strikingly different behavior than any other MOO, MUD or LMS. The rest of the sections of this paper discuss these features that adapt ProgrammingLand to the task of educating students.

MOOs and their predecessors, MUDs, use a telnet protocol to communicate. The user types commands and the system responds with lines of message text. The browser interface retains the ability to use a command-and-response type of interface, which is needed in some instances. However, the basic navigation inside the MOO is accomplished with a point and click interface as is the initiation of several other tasks. This mechanism also allows the use of graphic displays where that is helpful. Figure 1 shows a typical display of an exhibit.

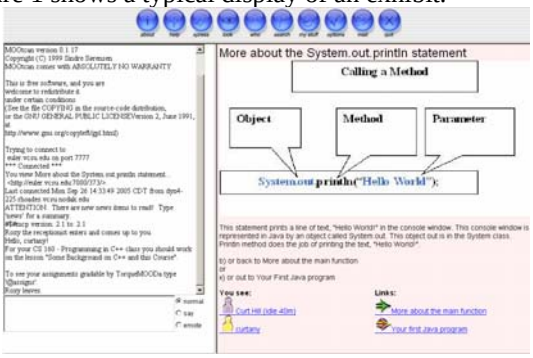

Figure 1. An exhibit displayed in a browser

Figure 1 shows a web browser window with four parts. The upper part display several buttons for common tasks. The right hand pane displays the exhibit text as well as hot-links to other rooms. In this display a small graphic is over the text. The icons for the viewing student as well as another student are at the left side of this pane and two links to other rooms are visible at the right side of the pane.

The bottom left hand pane shown in Figure 1 is where any command may be typed. Above that is a place for other types of message display. A student may use these two panes to chat with any other student in the same room. Interactive objects such as a code machine display their information and receive commands in these two panes of the browser. In contrast to this, the slide show object will use the right hand pane.

A MOO requires two pieces to properly function. The first is the server software which is freely available and the second is the core, which is the data that determines the arrangement of the MOO. ProgrammingLand is based upon a core from the enCore[2] project. The typical core should have 60 to 250 initial objects. This is typically built up using the MOO itself until the system is usable. ProgrammingLand currently has about 13,000 objects.

3. Lessons

ProgrammingLand has a larger organization superimposed on the room and exit structure mandated by MOO technology. Sections of rooms are structured into lessons. A ProgrammingLand lesson may have any of the characteristics of a lesson in the more traditional sense. A lesson may have any or all of the following: pre-requisite lessons, subordinate lessons, an introduction that motivates the student as to why this topic is important, content material that may be concise or detailed, exercises that reinforce the content material, tests to prove mastery and assignments that require completion outside of the system.

The system routinely records each exhibit that a student has visited, each lesson that has been completed, as well as the objects that the student has exercised. This information is stored on the student's history for later use.

Each lesson has a set of requirements that must be satisfied in order for the lesson to be completed. The past accomplishments of the student are compared with these requirements to see if the student has satisfied the lesson.

A lesson requirement is a two level list of goals or events. A goal is as simple as having visited a room or having viewed a slide show, to as complicated as having completed another lesson. A typical lesson has one to three sets of requirements. Satisfying the lesson is the process of achieving all the goals of any one of these sets.

There is one more variation in the requirements structure of lessons. Just as different instructors may require different readings of the same textbook, different instructors may require different events to complete a lesson.

A special type of room named a lesson room is the usual entrance into the cluster of rooms that comprise a lesson. The lesson room is linked to the requirement and lists all the rooms that are contained within it. The first time a student enters the lesson room, the requirements are shown. In any subsequent visit the student may use a command to display the requirements again, but this time any that are completed will be shown as such.

When a student completes a lesson, that event is appended to the student object. The lesson also notifies a dispatcher object that this particular student has completed this lesson. The dispatcher looks up the lesson in a list of lessons that have out of MOO assignments connected with them. If the lesson has such an assignment, an agent known as a roving goalie, appears to the student and tells the student the assignment. This description is also part of the student object so that the student may consult it at any later time. Because the lesson structure has most lessons subordinate to another lesson, most lessons do *not* have an assignment. The assignment is typically a programming assignment, but may be many other things as well, such as a worksheet to complete and hand in to the instructor.

The roving goalie may contain only one assignment or it may reference a list of equivalent assignments. These

are given in round robin fashion to the students as they complete the lesson. This diminishes the likelihood or success of cheating, since two friends will not usually receive the same assignment.

An instructor organizes the course with a course object. This object is the glue that holds the entire course together. Students are added to the course, which links the course to the student and the student to the course. The course also contains a list of the high level lessons, usually the ones with an assignment. The course also links to the requirements, allowing customized requirements. When a lesson is completed it notifies the dispatcher object. It does so by finding from the student the course and then from the course the correct dispatcher. The course object also points at an optional gradebook.

The course object allows a student to see where he or she is and should be, by displaying the lessons in the course and which have been completed. It also allows the instructor to monitor the progress of the students through the lessons. It is the information that is stored in the course and students that enables the software agents to be of assistance.

The course and lesson structure provides a nice advantage to ProgrammingLand over a lecture and textbook based class. Assignments are predicated on completing out of class activity inside the MOO. An instructor may never be confident that the student assimilated the material, but there is the guarantee that the student visited the needed exhibits before they received the lesson's assignments.

4. Interactive Objects

Reading a textbook, attending a lecture or browsing a ProgrammingLand exhibit is a passive learning exercise. Without some additional motivation this is not likely to produce the objectives an instructor will desire. An assignment will go a long way towards that desire and many a student will go back to their notes or textbook when a problem arises in the textbook. A good classroom instructor will combat this with in-class exercises and smaller pieces of homework before the larger assignment. ProgrammingLand also contains interactive objects of various sorts, which are there to engage active learning in the student. The earliest form of this is the code machine, but there are others as well.

The code machine is designed to demonstrate a short piece of programming code. It may display the code segment, with or without line numbers. The code without line numbers may be copied from the browser window and pasted into an editor. The code may also be explained or traced. In an explanation, the code is shown one line at a time followed by an explanation of why it is present and what it does. Since an explanation is longer than a simple display, the student needs to enter a command to go on to the next line and its description. A trace is a simulated execution of the code. Like the explanation it proceeds line by line with the student signaling their readiness to go on to the next line. An explanation always proceeds from the first line to the last in a sequential fashion, regardless of whether the line is executable or not. A trace cannot work that way since it is to simulate execution.

A trace is a simulated execution, thus ProgrammingLand does not in any sense interpret the code. It is preprogrammed into the code machine and always produces the same trace. However, the code machine trace has several useful features to enhance the educational goal. The simulation maintains the values of variables that may be used in the code segment. When a statement with a side-effect occurs the trace adds comments about the change in variables. The student may always ask the state of the current variables and the code machine will display them. The student does not ask about a single variable, the code machine always shows every variable, for code machine traces are typically small with few variables.

The simulation must maintain two important facts about an execution that may not be obvious to beginning students: the next line to be executed and the values that variables will receive. Thus the code machine may force the student into applying what he or she is supposed to know. When a flow of control statement is encountered the code machine may ask the student what the line number of the next line to be executed. It may also ask the new value to be assigned to a variable. Prior to either of these questions it will show the current values of the variables, then it is up to the student to decide what will happen next. If they are correct they are affirmed and if not the correct answer is shown.

The code machine is driven by a simple script that reflects the code at hand. The questions are optional, code machines encountered early typically do not have any questions. When a student has more experience with this object then they may be asked questions to force an active learning mode. The completion of either a trace or explanation may be made a requirement of a lesson, although the typical lesson has more code machines than is required by the lesson.

The practice and syntax test are other objects that require some thinking from the students. Both are based upon a table-driven parser[3] of a very simple construct. The student using the syntax test is given a statement or expression and asked to state whether it is correct or incorrect. If it is incorrect and the student answers correctly then he or she must enter the correction. The parser then determines if the result conforms to the syntax that is being considered. The student is awarded a point for each correct answer, thus the syntax test is a small completely automated quiz on simple syntax.

A syntax test may only be taken once by any given student. Therefore, near each room containing a syntax test is a room containing a practice object. This practice object uses the same parser table as the corresponding syntax test. The practice object does not ask questions or award points. Instead it merely pronounces whether the construct is properly formed or not. The student may then practice the skills needed for the syntax test, without the need of leaving the MOO or wrapping a program around the statement or construct in question.

There are other kinds of interactive objects as well. A slide show displays a sequence of HTML pages, usually with graphics, that make clear some topic. A typical show is the insertion of a item into a linked list. This is often best shown graphically and then explained as code. Others include the ring toss game, tutor robots and the history jukebox[4].

5. Agents

Students in college need no instruction in the use of a textbook, but ProgrammingLand is different from a textbook and sometimes students have problem. One issue is known as the short horizon problem. Once in an exhibit, only those rooms that are one exit are away are visible. Thus a student that has moved into a lesson may not be able to easily find their way back to known territory. Also, a student may not be able to find one of the requirements for a lesson so as to complete that and move on.

There have been two long standing solutions to this problem. First, exits that lead toward the entrance of the MOO have an icon with a different color to emphasize them. There is a also a series of web pages called the lesson map. This consists of a table of contents and index page, both of which have a link to a page for each lesson. The page for the lesson itself shows the shortest path to the lesson room and then details the subordinate exhibits. Despite this students were still getting lost or taking too long to find the requirements needed. Therefore two new software agents were implemented to help students make the most of their online experience. These agents are the lost and aimless agent.

Both agents have access to the information that makes it possible to determine if a student is lost or not making progress towards their current goals. There are two types of rooms that notify both of these agents of a student passing through, the monitor and lesson room. Any time a student enters these types of rooms the student is checked for lostness and aimlessness. If they are determined to be either of these the appropriate agent is sent to lend assistance.

When either of the agents is sent, it enters the room in the same way as another student or a course support staff. The agent then states that it appears the student is lost or not making progress towards a lesson and asks if the student wants to be moved to an appropriate room. The student may accept or decline.

The basic idea of ProgrammingLand is to be learner-centered. Thus if a student wants to explore areas that are not immediately pertinent, that is quite acceptable. Once a student declines an invitation to be moved to a more appropriate room, the agent will disregard any indications of lostness for a fixed constant, currently set at 10. Thus, if they decline they will not be asked again for some time, so that the agent will not be annoying. If they accept they are moved immediately to the new room.

A student is considered to be lost if he or she is in a lesson that is neither one of the completed lessons or the next immediate major lesson. The lost agent determines lostness by looking at the history of the student as well as the course of the student. The course references the major lessons and the major lesson refer to the subordinate lessons through the requirements.

Aimlessness is somewhat harder to determine. A series of progress objects is attached to the student object. One such object exists for each lesson that the student has started but not finished. Each visit to the monitor or lesson room causes the agent to reassess the progress. If after several visits there is no change then the student is considered aimless and the agent is dispatched to lend assistance.

When a student logs into the system an agent appears immediately and indicates a room that should be visited. This is generated by the same techniques that lost and aimless agents use to find a room for the student to visit. Unlike those two, this agent only speaks and does not move the student.

A quiz agent also exists to attempt to move a more experienced student through a lesson. If a student leaves a lesson, having completed all the requirements, except the visitation of simple rooms, then the quiz agent is dispatched. The quiz agent approaches the student and offers to give a quiz to prove mastery of the material. If the student accepts, the student is moved to a quiz room and the quiz is generated and given.

A quiz of this type is a five question multiple choice style test. If the student answers four or more correctly the lesson is complete. However, they are only offered a quiz for a particular lesson twice, should the quiz not be passed.

A grading agent[5] also exists, that is inspired by previous grading programs[6]. Strictly speaking this is not a MOO entity, but a stand-alone program that communicates with ProgrammingLand. When a student completes a programming assignment received from the MOO, the program, TorqueMOODa is executed. It contacts ProgrammingLand and displays a list of assigned programs. The student chooses one and this prompts TorqueMOODa to download a script from the MOO and exercise the executable produced by the student. The results are then uploaded to ProgrammingLand and stored on the student's object. Currently, TorqueMOODa is used not for the grading of the student's programs but as the last test for the program before it is turned in.

6. Learning Management Facilities

ProgrammingLand has a few of the features of a Learning Management System. Like any MOO it provides chat facilities in every room. It may be used to administer tests and surveys. Survey data is captured in the gradebook without interpretation, but a test may be partially graded.

A test consists of two parts, a part that may be graded by the MOO and a part that requires a person to grade. The former contains multiple-choice or true-false style questions. ProgrammingLand may grade these and store the result in the gradebook. Like a quiz question, a test question consists of one or more true answers and several false answers. The MOO randomly selects one right answer and up to four wrong answers and then presents

them in random order to the student. Those questions that require a more complicated answer are stored until the instructor may examine them and enter the score into the course gradebook.

The gradebook connects to the course object and records scores and computes a letter grade based on a scheme determined by the instructor. At any time a student may discover their own grade, while the instructor may observe all the grades or export the entries in a form amenable to import in a spreadsheet.

7. Conclusions and Future Work

ProgrammingLand has shown itself a functional resource for introductory programming classes. There seems to be no disadvantage in using it instead of a conventional textbook. The students appreciate that they do not have to buy yet another expensive book as well as the integration of assignments and tests.

There is a price to be paid for this resource and this is the student's learning curve as they adapt to the ProgrammingLand system. Generally, a small amount of class time is needed to orient the students into its use. Studies have shown this cost is offset by the advantages. Efforts to use the system without integrating it into the entire course have largely proven to be ineffective[4].

ProgrammingLand and associated software is covered by GPL, so may be freely acquired. Interested parties should contact the authors.

8. Acknowlegments

The ProgrammingLand MOOseum project has been supported by ND-EPSCoR through the FLARE program under EPS-9874802 and the National Science Foundation under Grant EIA-0313154.

9. References

[1] 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)

[2] Holmevik, Jan Rune, and Cynthia Haynes. enCore, Open Source MOO project. http://lingua.utdallas.edu/encore/index.html Date accessed 29 September 2005.

[3] Wetherell, Charles and Alfred Shannon (1981). LR- Automatic Parser Generator and LR(1) Parser. IEEE Transactions on Software Engineering, vol. 7, no. 3, May 1981, pp. 274-278.

[4] Hill, Curt, ProgrammingLand, An Automated System for Computer Science Education. PhD Dissertation, North Dakota State University, Fargo, ND, 2005.

[5] Hill, Curt, Brian M. Slator, Lisa M. Daniels. The Grader in ProgrammingLand. In *Proceedings of SIGSE 2005, St. Louis MO.*, (March. 2005).

[6] Foubister, S.P., G.J. Michaelson, and N. Tomes (1997). Automatic assessment of elementary standard ML programs using Ceilidh, *Journal of Computer Assisted Learning*. 13(2), June, pp. 99-108.