Progress towards Automatic Grading of Programs in ProgrammingLand

Curt Hill

Mathematics and Computer Science Department

Valley City State University

101 College St SW

Valley City, ND 58072

701 845-7103

Curt.Hill@vcsu.edu

Abstract:

An automatic program grader that is used in conjunction with the ProgrammingLand MOO is described. ProgrammingLand is an online resource for instruction in programming and Computer Science. As students use ProgrammingLand their progress is recorded and they receive assignments. Some of these assignments may be graded by the TorqueMOODa program.

TorqueMOODa starts an executable program and controls it as if it were the user. It may feed the program with input values and examine the output values or either a console program or a graphical user interface. TorqueMOODa acts based upon a script which it downloads from the MOO and some user inputs. When it is completed it records the results in the MOO.

This paper shows the usage of the ProgrammingLand MOO and TorqueMOODa from a student's perspective with additional comments on the script format and capabilities.

Introduction.

The ProgrammingLand MOOseum [1] is an online resource for Computer Science instruction. It is currently being used at Valley City State University as a required component of a classroom courses and also as the support for a fully online distance education course. In the classroom version of CS 1 it replaces the textbook, but serves as supplement in the subsequent classes. Among other things ProgrammingLand makes assignments to the students. In the fall semester of 2003 a new component was added, TorqueMOODa, the grader of programs.

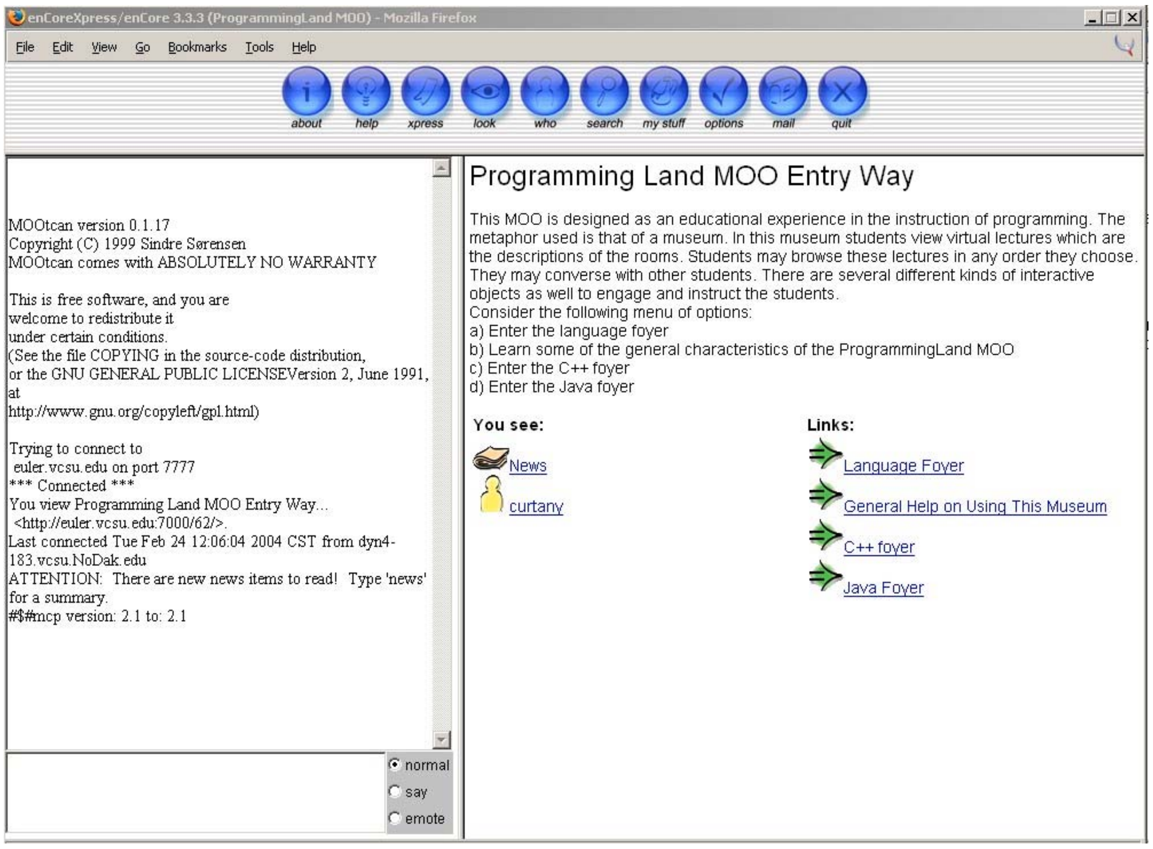
This project is part of the World Wide Web Instructional Committee (WWWIC) of North Dakota State University. The WWWIC program for designing and developing educational media is to deploy teaching systems that share critical assumptions and technologies (e.g. LambdaMOO; [2], [3]. Interesting projects include the Geology Explorer [4] and the Virtual Cell [5]. See also [6] for an overview paper.

This paper will briefly consider the student usage of ProgrammingLand in order to receive the assignment. It will then follow the student as they use TorqueMOODa to grade this program and report the results back to ProgrammingLand. The process of developing a TorqueMOODa script will be considered. Finally some results will be discussed.

Using ProgrammingLand.

The typical student uses ProgrammingLand by starting a Java enabled browser and connecting to the museum. There are numerous MOO clients available, but the one most commonly used at VCSU is the enCore Xpress[7] client. After the student has logged into the system they will see an image similar to that displayed in Figure 1.

The content of the museum is organized into clusters of exhibits termed lessons. A student browses through these exhibits and reads the content in approximately the same way as on a web site. They may also interact with several types of objects that enhance their experience. A lesson has a set of requirements, which may include visiting an exhibit, completing a subordinate or prerequisite lesson, completion of an interactive object. When the student completes all the lesson requirements, this event is recorded. The student has several commands available that help them keep track of where they are and what they should do next. The @course command would be typed in the lower left hand panel of the client shown in Figure 1. This will show each of the lessons that are required for the course, their roving goalies and whether they are completed. Figure 2 shows this display for our student.


enCoreXpress/enCore 3.3.3 (ProgrammingLand MOO) - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
about help xpress look who search my stuff options mail quit
normal
say
emote

```
MOOtcan version 0.1.17
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 Programming Land MOO Entry Way...
 <http://euler.vcsu.edu:7000/62/>.
Last connected Tue Feb 24 12:06:04 2004 CST from dyn4-
183.vcsu.NoDak.edu
ATTENTION: There are new news items to read! Type 'news'
for a summary.
#$#mcp version: 2.1 to: 2.1
```

Programming Land MOO Entry Way

This MOO is designed as an educational experience in the instruction of programming. The metaphor used is that of a museum. In this museum students view virtual lectures which are the descriptions of the rooms. Students may browse these lectures in any order they choose. They may converse with other students. There are several different kinds of interactive objects as well to engage and instruct the students.
Consider the following menu of options:
a) Enter the language foyer
b) Learn some of the general characteristics of the ProgrammingLand MOO
c) Enter the C++ foyer
d) Enter the Java foyer

You see:	Links:
News	Language Foyer
curtany	General Help on Using This Museum
	C++ foyer
	Java Foyer

Figure 1: Initial display in ProgrammingLand

```
Course: CS 160: C++ Programming
Course lessons:
 Background topics suitable for any programming language
    (Franklin)  - Completed.
 Some Background on C++ and this Course (Phyllis)  -
    Completed.
 Basics of C++ (Fred)  - Completed.
 The Idea of a Variable (Fernanda)  - Completed.
 Supplementary Lessons (Phoebe)  - Completed.
 The Compound Statement  - Completed.
 The if Statement (Frank)  - Completed.
 Switch Case Statement (Francis)
 The for statement (Frieda)
 ...
```

Figure 2: The results of the @course command.

Once a student has entered the correct lesson they may find the requirements for that lesson. The first time they enter the lesson the requirements are automatically shown to them in the middle left hand panel, displayed in Figure 1. If this is not the first time here, the student may explicitly request a display of the requirements by using the

@requirements command, which may be abbreviated to @req. Figure 3 shows this display for the example student.

```
Requirement 1
 Room: Why the Switch Case?  -- completed.
 Room: Syntax and Semantics of the Switch Case  --
   completed.
 Room: The Case Label and Following statements  --
   completed.
 Room: The Default Case  -- completed.
 Room: The Break Statement
 Room: Switch Examples  -- completed.
 Room: Final Notes on the Switch  -- completed.
 Room: Which should I use?  -- completed.
 Room: Variables in a Switch  -- completed.
 Room: Duplicate Case Labels in a Switch
```

Figure 3: The results of the @requirements command.

The student has clearly been to this lesson before this, since most of the requirements have been completed. This set of requirements has only one alternative and does not require anything but the visiting of rooms.

Certain lessons have an out of MOO programming assignment attached to them. Certain forms of practice are available in ProgrammingLand, but a full language system is not feasible. Therefore, an assignment system is implemented. Each time a lesson is completed, it notifies an object, called the dispatcher, that the student completed the lesson. The dispatcher looks up the lesson in a list of lessons, to see if an assignment does exist for it. If an assignment does exist then the dispatcher sends another object called a roving goalie. This roving goalie appears to the student like a player, but is merely a MOO object. The goalie contains a list of appropriate assignments. The assignments should be equivalent to one another in that they all use the same facilities of the language but are slightly different to reduce cheating.

From the student’s viewpoint the following actions occur. They enter the lesson room and are notified that they have completed it. (This happens with every lesson when it is completed.) In those lessons with an attached assignment a roving goalie appears after about two to three seconds. The visit of Francis is shown in Figure 4. The student may copy this text into a document, but the MOO records it as well. At any later time the student may execute the @showgoal command and receive the text in Figure 4, except for the first and last paragraph.

```
Francis enters the room and comes over to you and says:
Congratulations you have completed the Switch Case
lesson.
    Your next assignment is to write a portion of an e-
business application, using a Windows application. In
this program you will display a form with check boxes and
radio buttons as well as regular buttons, static text and
label text. This will simulate an electronic purchase
application.
    A generalized layout of this program may be seen at:
http://euler.vcsu.nodak.edu/francis.htm
    Each form should have a Calculate button, which is
used to compute the total price they are investigating,
an About button that displays a MessageBox with your name
and an Exit button which leaves the program. The options
that they choose will then determine the price that is
calculated. Each form should have labeled radio buttons
and check boxes. The results should be placed in a static
text box. All other information should be in labels.
    You will be selling cars. The base price is $20,000.
There should be a set of three radio buttons that
determine if the car is a 2 door (base price), 4 door
(extra $800), or convertable (extra $500). These radio
buttons should be in that order from top to bottom.
Check boxes should specify the addition of white wall
…
Francis leaves.
```

Figure 4: The arrival of Francis.

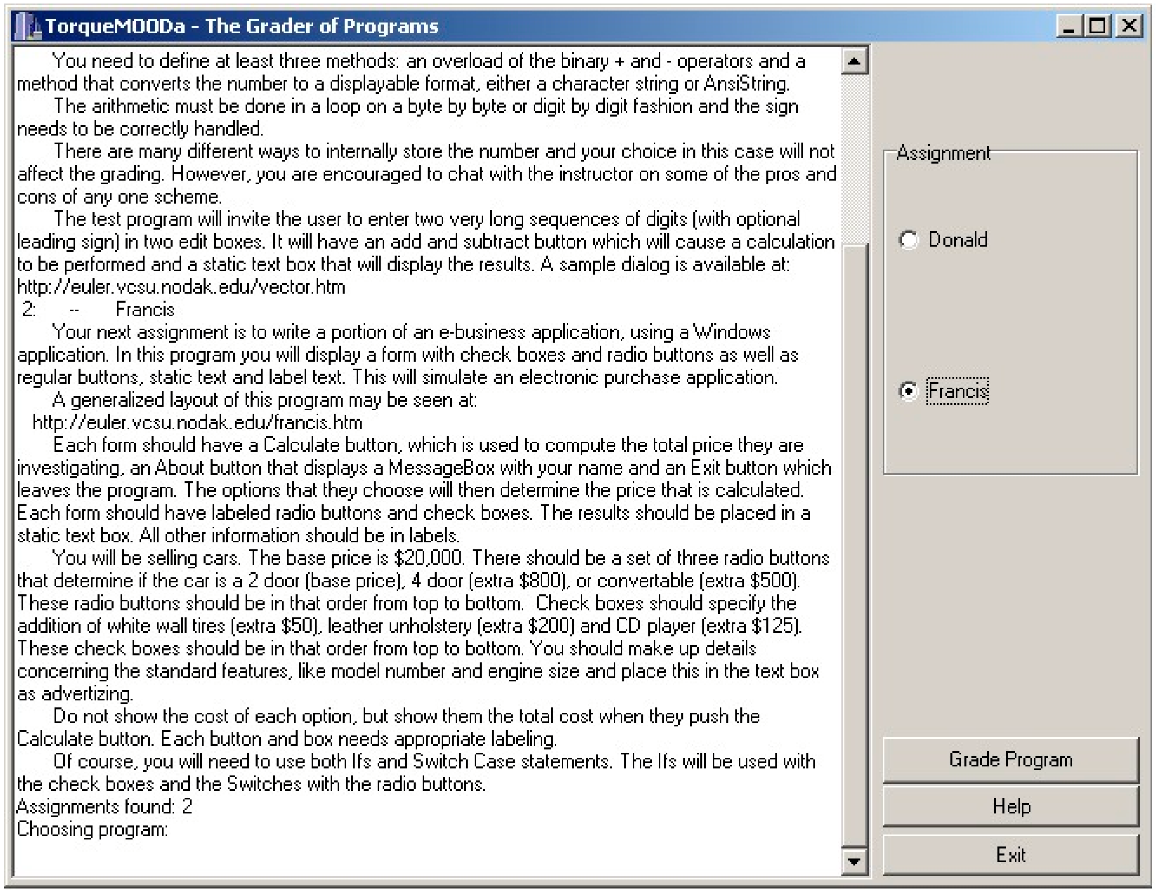
It may be argued that the ProgrammingLand MOOseum is superior to a textbook that contains the same material. It has always been problematic to enforce the reading of a textbook. This enforcement usually involves some form of testing, some of this testing quite unrelated to the stated goals of the course. This is not an issue in this course. If the students do not use ProgrammingLand they do not receive assignments. It is also rather easy for the instructor to monitor how students are progressing in their browsing of ProgrammingLand. The system keeps statistics for each student, on the number of times they have connected, the rooms they have visited and the lessons they have finished. Not surprisingly, the number of lessons completed correlates rather well with course performance.

The student may continue to explore the museum after having received an assignment, but if they have two ungraded assignments they will not be allowed into a new lesson room. Once students start work on their programs they may realize they did not understand the material as well they believed. They are never prevented from viewing any exhibits in lessons that they have started or finished.

In our observations of the sample student we will next advance in time to when the program is complete and it is graded by TorqueMOODa.

TorqueMOODa

TorqueMOODa is a stand-alone program that runs under Microsoft Windows®. (It is named after Tomás de Torquemada the first grand inquisitor of Spain and renowned for his cruelty.) The student downloads the executable for the program from a link on the class web site. When they execute TorqueMOODa it first logs them into the MOO. TorqueMOODa is not a general MOO client. However, it must download from the MOO the descriptions of the assignments and the appropriate script. Our example student would see the results shown in Figure 5 after the login process.

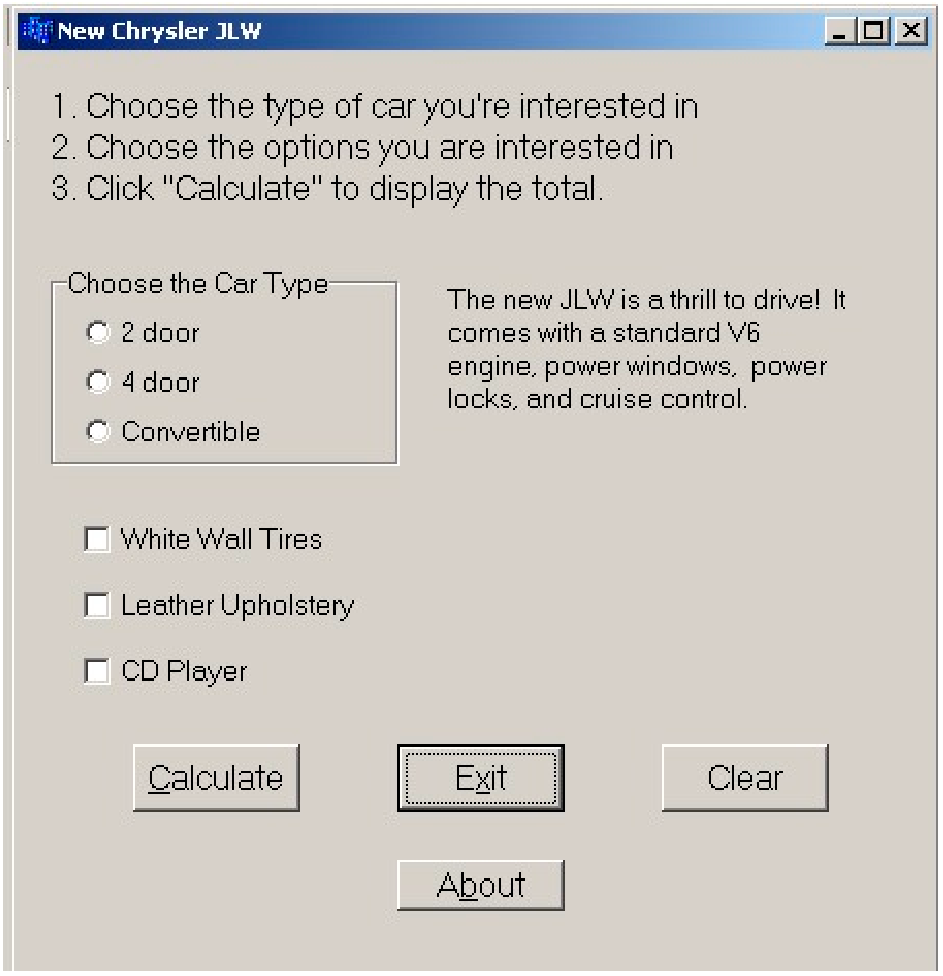


```
      You need to define at least three methods: an overload of the binary + and - operators and a
method that converts the number to a displayable format, either a character string or AnsiString.
      The arithmetic must be done in a loop on a byte by byte or digit by digit fashion and the sign
needs to be correctly handled.
      There are many different ways to internally store the number and your choice in this case will not
affect the grading. However, you are encouraged to chat with the instructor on some of the pros and
cons of any one scheme.
      The test program will invite the user to enter two very long sequences of digits (with optional
leading sign) in two edit boxes. It will have an add and subtract button which will cause a calculation
to be performed and a static text box that will display the results. A sample dialog is available at:
http://euler.vcsu.nodak.edu/vector.htm
 2:     --     Francis
      Your next assignment is to write a portion of an e-business application, using a Windows
application. In this program you will display a form with check boxes and radio buttons as well as
regular buttons, static text and label text. This will simulate an electronic purchase application.
      A generalized layout of this program may be seen at:
   http://euler.vcsu.nodak.edu/francis.htm
      Each form should have a Calculate button, which is used to compute the total price they are
investigating, an About button that displays a MessageBox with your name and an Exit button which
leaves the program. The options that they choose will then determine the price that is calculated.
Each form should have labeled radio buttons and check boxes. The results should be placed in a
static text box. All other information should be in labels.
      You will be selling cars. The base price is $20,000. There should be a set of three radio buttons
that determine if the car is a 2 door (base price), 4 door (extra $800), or convertable (extra $500).
These radio buttons should be in that order from top to bottom.  Check boxes should specify the
addition of white wall tires (extra $50), leather unholstery (extra $200) and CD player (extra $125).
These check boxes should be in that order from top to bottom. You should make up details
concerning the standard features, like model number and engine size and place this in the text box
as advertizing.
      Do not show the cost of each option, but show them the total cost when they push the
Calculate button. Each button and box needs appropriate labeling.
      Of course, you will need to use both Ifs and Switch Case statements. The Ifs will be used with
the check boxes and the Switches with the radio buttons.
Assignments found: 2
Choosing program:
```

Figure 5: TorqueMOODa offering a choice of assignments.

This student has two assignments pending, the first from Donald and a second from Francis. The descriptions of these assignments are given in the main part of the display with a radio button that allows the student to select which assignment to grade at this time.

When the student clicks the grade program button, a dialog box allows the student to find the executable. TorqueMOODa, for the most part, does not care what type of program it is provided it has a standard executable. Figure 6 shows the window that corresponds to the assignment given earlier by Francis.



1. Choose the type of car you're interested in
2. Choose the options you are interested in
3. Click "Calculate" to display the total.

The new JLW is a thrill to drive! It comes with a standard V6 engine, power windows, power locks, and cruise control.

Figure 6: The running program.

TorqueMOODa causes the selected program to be executed. The program has an idea of what types of controls should exist in the program. The specification required three radio buttons, three check boxes and three regular buttons. This program supplied four buttons, but the extra button is not an error. At best TorqueMOODa can only guess where the controls are located. Forcing the students to exactly mirror the look of the application may suppress their creativity, so is not a completely desirable alternative. Instead TorqueMOODa displays a list of the controls it must have and a mock window of the application. The user may then correct any of the guesses that were made about the location of item. The display of the desired controls is shown in Figure 7 and the mock window is shown in Figure 8.

The user is to connect the numbers given in the TorqueMOODa display with the guesses in the mock window. The mock window is constructed to have approximately the same size as the original, except for the extension on the side that contains the OK button. The display in the TorqueMOODa window gives a brief description of what the item is to indicate and what type it is. Thus for example item 1 is a radio button that indicates that the car is regular, that is neither a four-door or convertible.

```
Check these descriptions with the numbers in the mockup
     window.
 When the values are correct in the mockup click OK on
     that window for grading to proceed..
 Values of -1 indicate controls that are not connected to
     one of the following:
[1] Regular   << Radio Button >>
[2] Four door   << Radio Button >>
[3] Convertible   << Radio Button >>
[4] Whitewall   << Check Box >>
[5] Leather   << Check Box >>
[6] CD Player   << Check Box >>
[7] Answer   << Static Text >>
[8] Calculate   << Button >>
[9] About   << Button >>
[10] Exit   << Button >>
```

Figure 7: TorqueMOODa offering a choice of assignments.

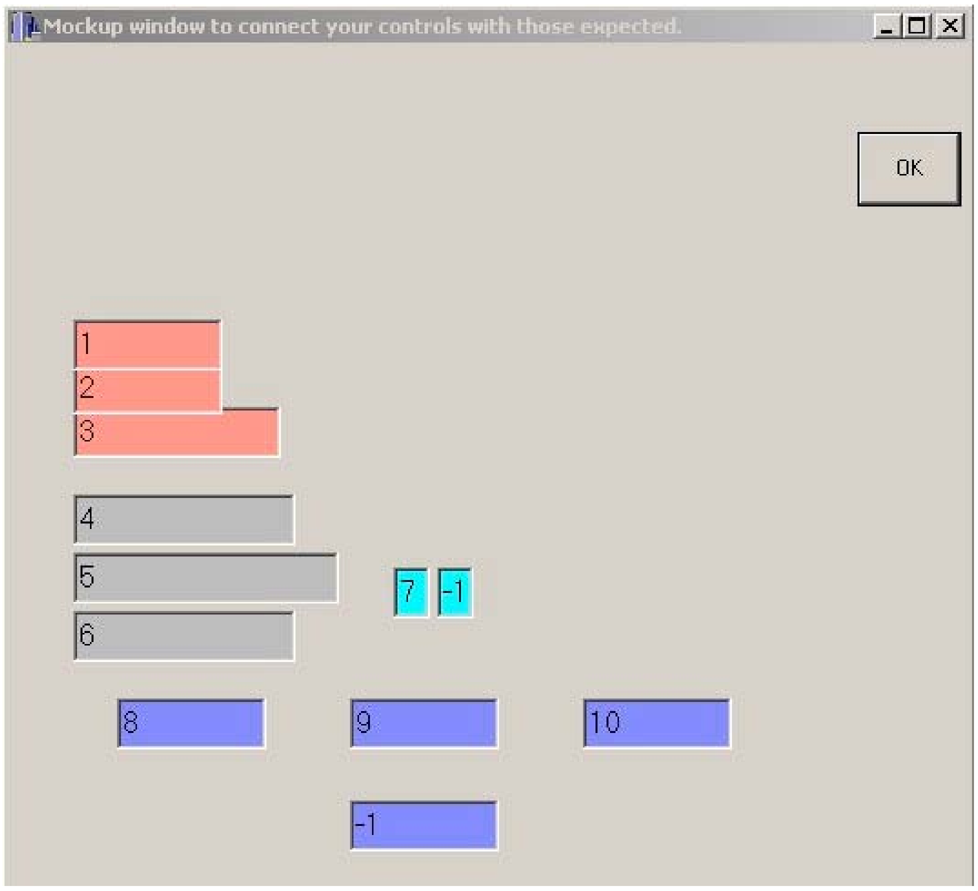


Figure 8: The mock window.

In this example there are several control errors. A control error is whenever TorqueMOODa guessed wrong on the placement of the controls. A control error will not cause a deduction of the student's score if it is corrected. The program guessed correctly on controls 1-6 and 8, but the rest are wrong. The mockup has 7 as where the vehicle price will be placed. Actually, this area only contains a dollar sign. The actual price goes next to it in the control labeled with a -1. TorqueMOODa puts a -1 in every control that it does not connect to an expected control. The About button also has a -1, because this student added an additional button, the Clear. Therefore the two results field and the three buttons would have to be corrected by the student, before the OK button on the mock window is clicked.

When the student makes changes and clicks OK on the mock window, it disappears and the program starts the grading process. In this phase TorqueMOODa may click or unclick radio buttons or checkboxes, enter data into edit boxes, read the contents of text fields and click buttons. It processes its script to test the various features of the program. As it does so it shows the results in its display area. Figure 9 shows this display.

```
Check plain car
 Earned 2/2, running score 2/2
Correct
Check 4 door with CD
 Earned 2/2, running score 4/4
Correct
Check loaded convertible
 Earned 2/2, running score 6/6
Correct
Final score: 6/6
Source code examination
 Earned 6/6, running score 12/12
Source code comment examination
 Earned 2/2, running score 14/14
Final Score: 14/14
Score recorded
You may now exit the program
```

Figure 9: The results.

The TorqueMOODa Script Language

TorqueMOODa is a generalized program to execute other programs as if the TorqueMOODa is the user. It may handle either interface paradigm, console or graphical user interface. In a console program it may write data into the test programs input and read the program's output and examine it. In a GUI-style program it may place data in edit boxes, click radio buttons, check boxes or buttons, read and interpret data in an edit box or text box. The ability to do these things is specific to the Microsoft Windows platform, although there is an intent to make a Java and LINUX version of TorqueMOODa.

TorqueMOODa is generalized, but requires a specific script to examine a particular program. This script should indicate the interface paradigm of the program, the data to give to the program, the expected results to receive, and the points to assign to types of answers. This paper will not give an extended description of the script language, but will instead explain the particular script used in the example. Figure 10 shows this example script. The line numbers have been added for ease of reference and are not part of the script.

The script is actually compiled into a more compact and interpretable form. This form is stored on the MOO for later download. The TorqueMOODa Script Compiler (tmdc) is based upon a table generated parser. It loads in the current parser tables and then parses the input script. The recognition of a production causes data to be stored or a line of the output script to be produced. The compiled script is about 60% of the size of the original.

The output script is also ASCII text, but somewhat less people readable. Each line represents one instruction to TorqueMOODa. The operation is represented by a single letter. The rest of the line represents the parameters to the operation.

The script may have a passing resemblance to C++, such as the one line comment, the assignment statement form, the frequent use of semicolons, and the braces for a compound statement. However, the script is much more restrictive due to its limited purpose.

The first thing to notice in Figure 10 is the use of the Control function. The control function is used like a variable declaration in the script. The first parameter determines the type of the control. Lines 2 - 4 show radio buttons, 5 – 7 check boxes, 8 a static text area, and 9 – 11 buttons. This example does not show a menu control. TorqueMOODa determines if the program is a console or window program based upon controls. The presence of any window control forces that paradigm. A *variable* type as the first parameter indicates a console style program.

The name following the type in the Control is the name of the variable. This name will be used in the rest of the script. The value that the control contains may be examined or changed depending on the control. For example the assignment on line 18 of a true to regular is setting the radio button to a clicked value.

The parenthesized pair of numbers following the type and name of a Control is the approximate location. This location and the control types allow TorqueMOODa to make an initial connection between the actual programs controls and those given by the script.

Originally that was the only connection mechanism, but it proved too unreliable. The mock window has proven much more effective.

The final parameter of the Control function is a text description of what the control is supposed to indicate. This piece of text is used in the display pane when the mock window is shown, so that the student can see what is expected. Figure 7 shows this message display with the types of the controls.

```
 1 // The selling of items from Francis, this one is cars
 2 Control(radio,regular,(30,90),"Regular");
 3 Control(radio,fourdoor,(30,120),"Four door");
 4 Control(radio,convertible,(30,150),"Convertible");
 5 Control(check,whitewall,(20,200),"Whitewall");
 6 Control(check,leather,(20,230),"leather");
 7 Control(check,cd,(20,260),"CD Player");
 8 CONtrol(text,answer,(250,200),"Answer");
 9 Control(button,calc,(150,300),"Calculate");
10 Control(button,about,(250,300),"About");
11 Control(button,exit,(400,300),"Exit");
12 source("if","Must use an if statement.",2,0);
13 source("switch","Must use a switch case statement.",2,0);
14 source("MessageBox","Must use a message box for the about button.",2,0);
15 MapControls(user,0);
16 // Try something
17 Message("Check plain car");
18 regular = true;
19 whitewall = false;
20 leather = false;
21 cd = false;
22 execute(calc);
23 if(answer == currency(20000.00)) {
24   Score(2,2,"Correct");
25   }
26 else {
27   Score(0,2,"Incorrect price");
28   }
29 endif;
. . .
56 Comment(10,10000,2);
57 comment(5,9,1);
58 End(exit);
```

Figure 10: The Francis TorqueMOODa script.

Line 15 shows a MapControls function. This particular function forces the examination of a window. The reserved word user indicates that this will be a user designed window that requires the mock window to be generated. It is also possible that TorqueMOODa could be asked to verify the presence of a MessageBox, InputBox, or some form of Common Dialog Box. However, this script does not demonstrate any of these. The second parameter indicates the number of points to give the student if the window is correct. In this case the value is zero, since no grading can occur unless the window can be successfully mapped onto the controls that are given. The values are usually used for the predefined styles of dialogs.

Lines 12 – 14 describe things to find in the source code of the program. These may occur anywhere in the script but their effect is deferred until after the run-time grading is complete. This script on line 14, reflects an older way to check for a MessageBox, by finding the command in the source code. The source code processor does a simple scan of one or more source code files. It removes comments and then allows regular expression processing of what remains.

The second parameter of the source function is an error message to display if this feature is not found. The third parameter is the number of points to give, should this expression be found. The last parameter is the pool to which this function belongs. One of the requirements of Francis was that each program should use an *if* statement, a *switch* statement and a *MessageBox* call. The student was awarded two points for at least one occurrence of each. They did not receive two points for each if, but two points if any of the source files had an *if* statement.

All of these source examples were in pool zero, that is they were independent. In this script there were six points available in pool zero. A program that received all six must show one or more of each of the constructs. Pools other than zero allow alternative ways to satisfy a requirement.

If the assignment had been only to use a decision statement, then either an *if* or *switch* could have used. In that case the source function pool parameter would have been non-zero and the same for both the *if* and *switch*. In dealing with a pool only one of the source statements needs to be satisfied.

Line 18 indicates the beginning of the interaction between TorqueMOODa and the running program. However, before that come a Message function, which has the sole purpose of announcing something on the display. The first line of Figure 9 shows the only effect of the Message function, which is to inform the student of what it is currently testing.

Lines 18 – 21 set values for the first test. The script does not examine the original values of the radio buttons and check boxes, but it must set each before the first test. The initialization of the input values is followed by an Execute function. An Execute must have a button as a parameter. The effect is to click the button and wait for a short time for the program to respond. The default time to wait is one second, although the Wait function may be invoked to lengthen that time if the processing merits such a delay. The clicking of the calc button, then sets the stage for the first evaluation of results.

What is expected is that the static text area identified by the Control named answer now contains the calculated price. Therefore the value of that control is compared against the desired value of 20000.00. It is usually not that easy. The student may or may not have formatted the number before displaying it. The currency function strips the results of dollar signs and commas before the comparison. In this way the following are considered equivalent: $20,000.00 or 20,000.00 or 20000.00.

Should that comparison be true, then compound statement containing the first Score function is executed. Strictly speaking the compound statement is not needed. An *if* may contain one statement and the Score would be that. However, there is no harm in wrapping a single statement in a compound statement.

The Score function is the main means of accumulating scores in TorqueMOODa. It contains three parameters. The first parameter is the score to give to this part of the program. The second parameter is the number of points possible. TorqueMOODa maintains two scores for the program, the points earned and the points available. The first two parameters are added to these two values respectively. The third parameter is a string to be displayed. Figure 9 shows that this program took this path and earned two of two points. Had the value been incorrect, then line 24 would have been skipped in favor of 27.

The *if* statement of the script is somewhat different than that of C in at least three respects. First, only an equality comparison is allowed. Second, one of the items to be compared must be a Control. Third, the *if* statement must be terminated by the reserved word *endif*, which removes the common ambiguity from the grammar.

Lines 30 – 55 were omitted as they show two more test cases similar to the prior one. The number of points possible in TorqueMOODa is determined only by the paths through the script. All possible paths through the script should yield the same number of available points, although neither TorqueMOODa, nor the script compiler enforces that. There are no loops in the script language so writing scripts that conform to this assertion is not hard. Most scripts have the same basic form as this one: establish controls, do one or more test cases, finish up. Each test case has the following form: establish values, execute, examine results in one or more ifs. Since these ifs may be nested, more than two possible scores may be earned in a single test case.

The Comment functions in lines 56 and 57 are part of the source code scanning system like the previously encountered Source functions. The locations of these in the script file are irrelevant, the source is always scanned last. The source code for the program is only scanned if one or more Source or Comment statements is encountered in the script. Each comment has three parameters, the first two are the range of comment words and the last is a score. The source code scanner removes comments, but counts words inside comments. A word is one or more letters that terminates with a blank, comma or period. This is a rather coarse definition, but adequate for this purpose. In this example, they get two points if they have more than nine words inside comments, one point for five to nine words and no points otherwise. This program makes no attempt to determine if the comments make any sense, it merely counts the words. The TorqueMOODa compiler verifies that there is no overlap in the number of words in Comment functions.

The final statement is the End statement. It is essentially the same as the Execute, in that it expects to click a button and have some effect. However, the effect it expects is for the program to terminate. There is a deduction if the program does not end on its own. A console program would not need an End, it would normally end as soon as the final input had been read and the final output displayed.

Conclusions and Future Work.

TorqueMOODa is still at an early stage, so no serious comparison between its results and a human grader have yet been conducted. The first versions were available in the Fall semester of 2003, though improvement continues. Correlations of scores produced by early versions of TorqueMOODa and human graders were low, typically in the 0.1 to 0.3 range. A later edition of TorqueMOODa and the Francis programs produced a more respectable 0.59 correlation. Even so there appear to be several problems remaining.
In general TorqueMOODa is very sensitive to the specifications. Strict conformance to the specification is generally a good thing. However, in the first programming course a little freedom in this area tends to make the students more creative and increases their whole enthusiasm for the whole field. Dampening this enthusiasm here may restrict the number who will continue. TorqueMOODa can be very unforgiving to a student who makes a program more interesting than required. The example given in the paper was not penalized, but they did not alter the basic user interface, only added it to it. In a console program this would have resulted in a very poor score.
In general console programs are more difficult for TorqueMOODa, because it can be thrown off by lots of harmless things. These include varying numbers of initial announcement output lines, numeric output values that are examples rather than results, extra questions on input, and different orders of values to be input. Also finding the desired value in a line of output is problematic. Regular expression processing allows finding numeric values in a line of text, but the simple inversion of the order of two values is difficult to resolve. The corresponding problem in GUI programs was largely settled by the mock window approach. This requires a console program to be very tightly specified, even down to the order of inputs.

GUI programs have fewer problems but improvement is still needed. TorqueMOODa does not yet realize the equivalence between certain types of controls. For example a student may display the output value in a label area, a static text area, or an edit box. A human grader will have no problem with either of these. However, a label cannot be seen by TorqueMOODa at all. If it expects either a static text area or an edit box it will not recognize the other as the source of a value. If the student places extra text in a control expecting a numeric value the pattern matching fails. Even small control problems yield very poor scores, then a human has to correct the program so that it will grade properly. Furthermore, no mechanism for sampling a multi-line edit box has been yet adopted, nor have menus been tested.

Acknowledgements.

My thanks to Justin Bearinger for the original creation of many TorqueMOODa scripts, including this one for Francis and to Jacqueline Wolf for allowing me to use her program as the example in this paper.

The ProgrammingLand MOOseum project has been supported by ND-EPSCoR through the FLARE program under EPS-9874802 and is currently supported by National Science Foundation Grant EIA-0313154. The NDSU Worldwide Web Instructional Committee (WWWIC) research is currently supported by funding from the National Science Foundation under grants EIA-0086142, and from the US Department of Education under FIPSE grant P116B000734.

References.

[1] Hill, Curt (2001). Evolution of ProgrammingLand. Proceedings of the Midwest Instructional and Computing Symposium (MICS 2001), April 2001, Cedar Falls IA.
[2] 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)
[3] Haynes, Cynthia , and Jan Rune Holmevik, eds (1997), High Wired: On the Design, Use and Theory of Educational MOOs. University of Michigan.
[4] Saini-Eidukat, Bernhardt., Donald P. Schwert, and Brian M. Slator (2002). Geology Explorer: Virtual geologic mapping and interpretation. Computers and Geolsciences. 28(10), pp. 1167-1176.
[5] 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. pp.1444-1445
[6] Slator, Brian M., Paul Juell, Phil McClean, Bernhardt Saini-Eidukat, Donald Schwert, Alan White, Curt Hill (1999). Virtual Environments for Education. The Journal of Network and Computer Applications. 22(4), pp.161-174. Academic Press.
[7] Holmevik, Jan Rune, and Cynthia Haynes (2004). enCore, Open Source MOO project. http://lingua.utdallas.edu/encore/index.html Date accessed 11 February 2004.