TclTutor

Clif Flynt

Noumena Corporation
9300 Fleming Rd
Dexter, MI 48130

Abstract

TclTutor is a Computer Aided Instruction application for learning the Tcl programming language.

This paper discusses the application's development and some of the technical details, along with lesson's learned and potential future work.

Introduction

TclTutor is a Computer Aided Instruction application for learning the Tcl programming language. The application is written in Tcl using only the Tk extension. It runs on any platform that supports Tk.

Since its introduction in 1995, it has gone through 3 major revisions and several minor bug fix or feature releases. It's been downloaded hundreds of thousands of times. The lessons have been translated into Portuguese and extracted as the basis for the www.tcl.tk online tutorial. It directly led to my getting a contract to write a book about Tcl/Tk and was instrumental in my being hired to teach at Grinnell College.

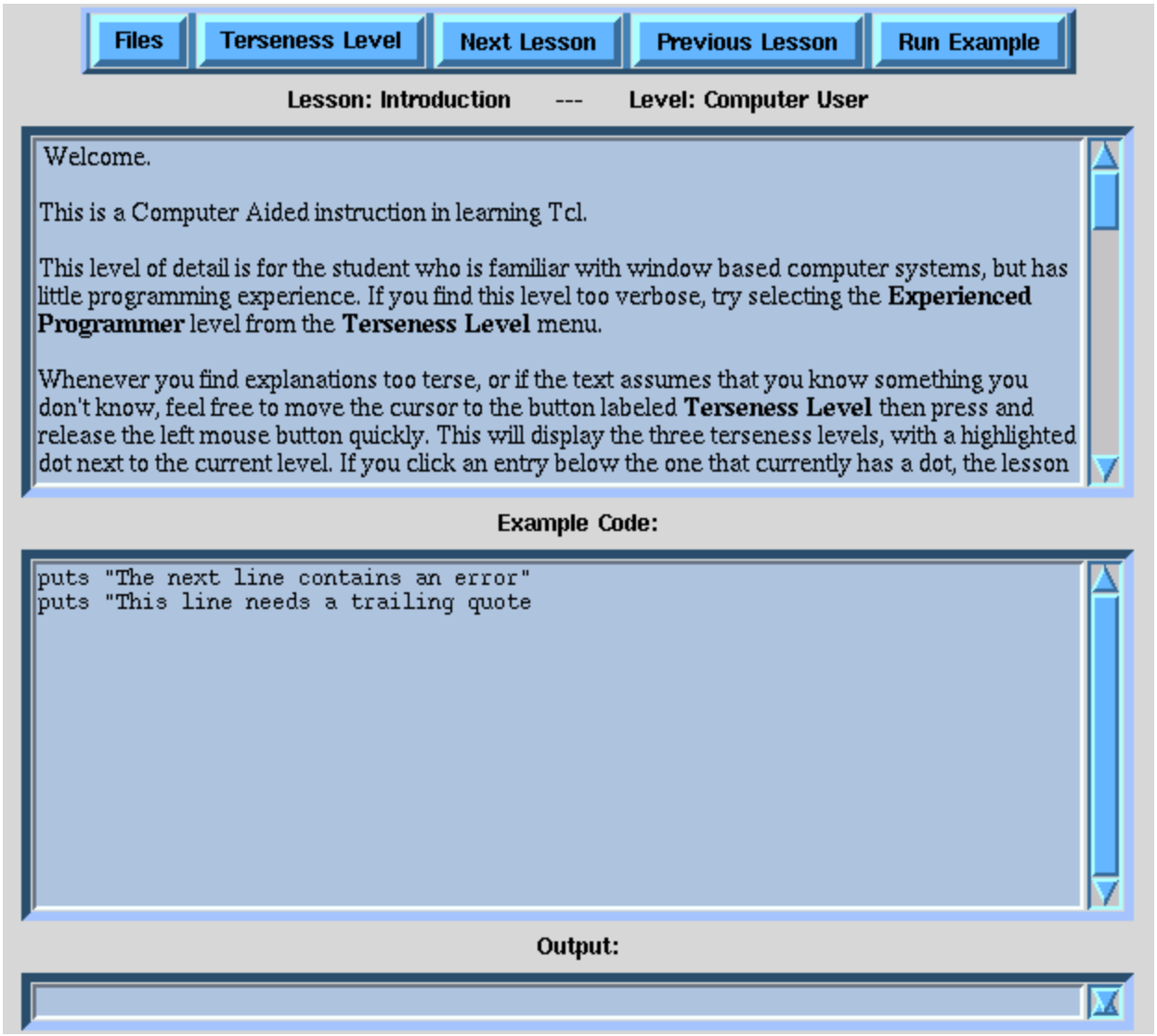
It's undoubtably the most successful application I've written in 30+ years as a professional computer programmer. It has several good features that were included by accident and some warts that nobody can miss.

History

The 1995 the Tcl Conference included a panel to discuss using Tcl and Tk for college level coursework. Two of the participants, Charles Crowley and Joseph Konstan commented that they needed a fast method to get students up to speed with Tcl and Tk in order to use Tk as a vehicle for teaching aspects of computer science. [Crowley95] Their comment inspired me to create the TclTutor package, as a solution to this problem.

I took a few notes on the back of the program and started coding as soon as I got home, much to my wife's dismay. The base engine was written in a few hours, the lessons, over several weeks.

The first released version of TclTutor looked like this:



When Tk was ported to the Microsoft Windows operating system, I ported TclTutor. This involved a fair amount of rework, since the original version TclTutor used `exec` command frequently. Code like `exec ls` was replaced with `glob`, while the `exec grep` constructs were replaced with Tcl procedures to examine files. The `file` command was introduced when Tcl was ported to Windows, and all the hardcoded paths needed to be replaced with `[file join ...]` constructs.

These were all mechanical changes, but they introduced me to a lot if programming styles that I'd been able to comfortably ignore until then.

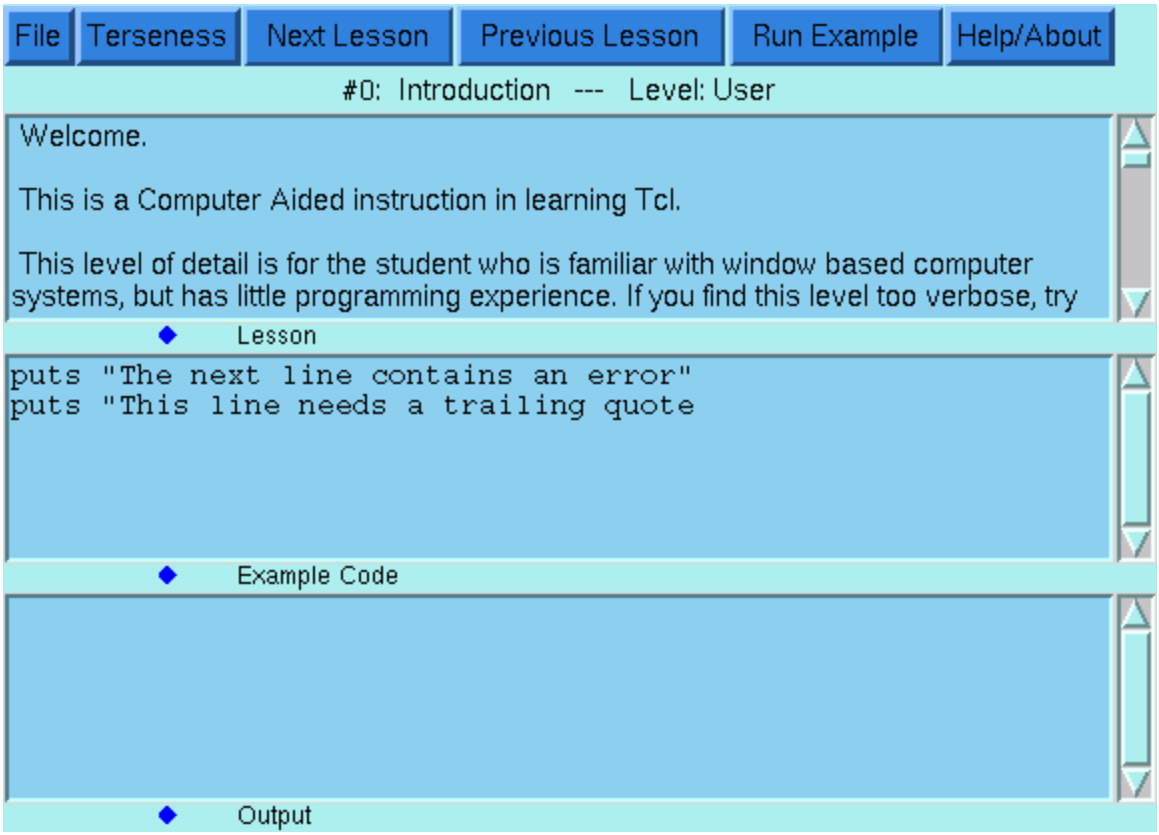
The TclTutor engine stayed fairly constant for a few years while I reworked installation kits that used InstallShield (with limited success). When application wrappers became available, I made TclTutor friendly with Jan Nijtman's original `wrap` program, then `Freewrap`, the TclPro wrapper, and finally, in 2002, TclTutor got ported to use a `StarPack`.

The biggest advantage of using the application wrappers was that this reduced the amount of email I recieved asking why TclTutor would not work and if they needed to install Tcl first.

The first major rework of the GUI came in 2000, when the `grid rowconfigure` command made it possible to easily make windows that were dynamicly resized under script

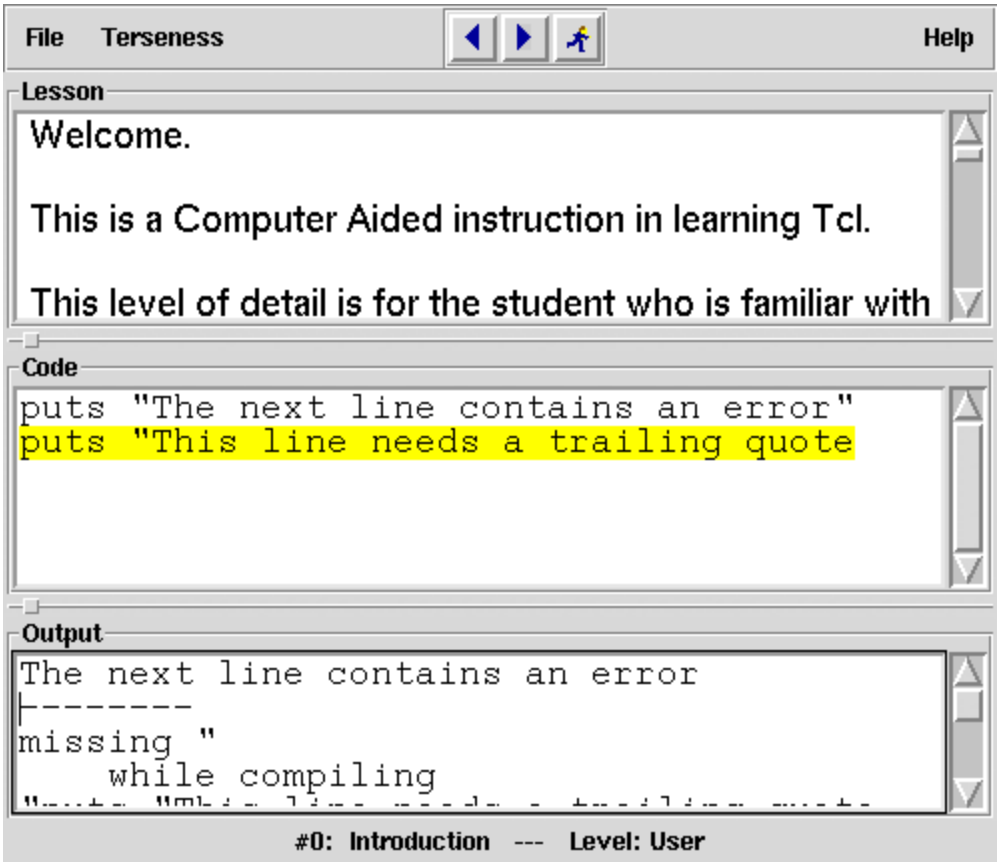
control. I wrote a paned window controller for TclTutor that would let users resize windows.

At that point, TclTutor looked like this:



The dark blue diamonds are the resizing controls. Not elegant, but it worked.

For the past 5 years, TclTutor has existed with this GUI. Finally, TclTutor is moving into the 21'st century with real paned windows, and the battleship grey coloring.



Design

The design and implementation of TclTutor has paralleled the evolution of our concepts of how Tcl can be used.

Originally, John Ousterhout conceived Tcl as an adjunct to an application written in a compiled language like "C". Later users saw Tcl as a better shell scripting tool that uses the `exec` command to invoke other applications to perform the actual work. Tcl is now perceived as an extensible interpreter and the common pattern is to write a complete application in Tcl, with small sections written in a compiled language.

The initial version of TclTutor followed the paradigm that Tcl was a better Bourne Shell with a nifty GUI tool. It was written in a single file and most of the code ran in the global scope. It made frequent use of the `exec` command to perform operations that were too slow or cumbersome for the 7.3 release of Tcl.

The single file version of TclTutor lasted until Steve Uhler [Uhler95] released `hmtllib.tcl` pure Tcl HTML rendering package. TclTutor was an early adopter of this package.

The early versions of TclTutor kept the application and all the related course and lesson files in a single directory. Later versions of TclTutor supported a separate lesson directory where all lesson files could be kept. The latest TclTutor is finally moving to a directory tree for the courses with each course in a separate subdirectory.

While new lessons have been added and the GUI has been updated, the basics of TclTutor haven't changed since the first version.

The two critical parts of TclTutor are:

- Multiple verbosity levels.

  The holy grail of Computer Aided Instruction has been to duplicate the way a good teacher will rephrase information to match the needs of the student.

  The three verbosity levels in TclTutor provide a simple way to give the beginners more information without boring the experienced users.

- 3 Windows - text, example and output.

  Another attribute of good CAI is to provide the student with immediate interaction and feedback.

  Tcl provides all the hooks to make an interactive window where the user can view the example, run it, see the results, edit and try again.

The lesson and course menus are generated at runtime. The application examines a given directory and builds the menus based on simple naming conventions. Each lesson will include a file with a `.cfg` suffix which contains some language specific settings and special procedures for operations like parsing error output. The lessons are named by the topic name, followed by a number, then a `.lsn` suffix.

The `.cfg` file allows TclTutor to be easily configured for teaching other material. Each course includes a course config file which defines the procedures that are used to run examples and report errors. Once the config file is constructed, the lesson files all follow the same format.

One advantage of this design is that once a course configuration file has been created, non-programmers can develop lessons. The primary released version of TclTutor have been for Tcl lessons, I've developed lessons for extension like Spirent/AdTech AxTcl. I've also created proof-of-concept lessons for perl, java and scheme as well variations like multiple choice tests. The Tk lesson set has been *in development* for several years, with varying amounts of attention.

Tcl/Tk as a platform for developing CAI applications.

The fact that TclTutor is written in Tcl provides an existence proof that Tcl/Tk is an adequate language for writing a CAI application. In fact, TclTutor barely scratches the surface of what can be done in this arena.

A good CAI development environment should:

Encourage the programmer to focus on CAI Development Tasks.
: The programming language should contain the constructs necessary for the task. If the language has inadequate tools, or imposes an inappropriate presentation paradigm, the programmers will spend more time dealing with the programming environment than creating the CAI program.[Bork87.1]

Support rapid prototyping.
: One of the most expensive phases of constructing a CAI program is the design-program-test-redesign phase. The programming environment should support mechanisms which shorten this phase.[Bork87.1]

Support easy modification of lesson units.
Teachers may request improvements after using the package with students. Changes in education theory and practice may mandate new material. The capability to easily modify lessons allows a package to remain current and useful. [Bork87.1] This implies separating the lesson materials from the presentation engine.

Support interaction and feedback.
Studies have found that immediate feedback increases learning. [Bork85] [Bork87.2][Owens92] The programming language should include constructs which support feedback mechanisms.

Support multimedia and hypertext.
Studies have found that different presentation methods (video, sound) improve student comprehension and retention. [Fox96] The programming language should include constructs for multimedia and hypertext. [Owens92] [Barron95]

Be portable.
Hardware and software platforms may differ from school to school, or even within a single school. A good CAI package must accommodate varying environments. [Bork87.1]

Be available at a reasonable cost.
Educational institutions, expecially those below the college level have limited budgets. [Bork87.1]

Tcl/Tk meets all of these requirements quite handily.

Technical Details

The TclTutor lesson engine is a relatively simple wish program containing a menu, task bar and 3 text windows.

The main features of TclTutor (three text windows and 3 levels of verbosity) have remained constant through 12 years of use and tweaking. The implementation of these features has evolved as the concept of how one should use a scripting language has evolved.

The largest code change was reworking the engine to support multiple platforms. This caused my paradigm for Tcl to shift from Tcl being a better shell language to Tcl being a high-level interpreted language.

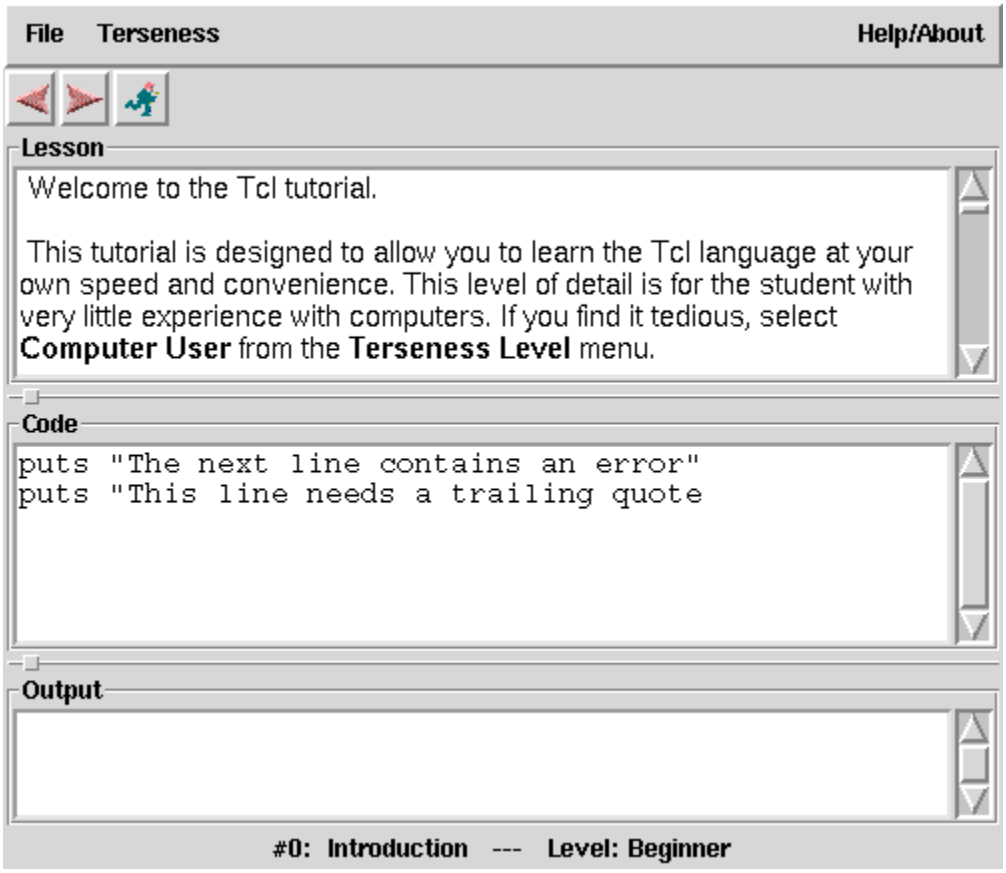
The first version of TclTutor was composed primarily of global scope code with only a few procedures. As TclTutor got reworked, the coding style moved from the script-style global scope code to a more structured style in which most of the code is in procedures and only a small amount of setup code is in global scope.

The most visible changes to TclTutor have been to the GUI. These reflect the evolution from the early X Windows/Motif philosophy of using interesting colors to the modern boring gray approach, and the evolution of the Tk Toolkit to support the new GUI standards.

The early versions of TclTutor tried to economize on desktop space by using a single

control bar that mixed `menubuttons` and normal buttons. By default the normal buttons and menubuttons were displayed with a different appearance, but the configuration options provided plenty of power for making these look identical enough to confuse any users.

An unreleased version of TclTutor looked like this:



This follows the modern practice of having a menu at the top, and a taskbar for buttons.

The taskbuttons allow students to step through the lessons and run examples. The most common applications a user does.

The menus allow a student to change courses, select a specific lesson, change verbosity levels or set various options like colors and font sizes.

Separate menu lines and taskbars make a lot of sense for applications like Open Office or Firebird where there are too many elements to fit on a single line. It looks silly to have a mostly empty menu line and then an even emptier taskbutton line.

Given the simplicity of the user interaction with TclTutor, I decided to stick with the single row of menu buttons and task buttons. To make it a bit easier to distinguish between these buttons, the menubuttons are flat relief with textual labels, while the taskbuttons are raised relief iconic labels.

The construction of the course and lesson selection menus is done on the fly. The first (unreleased) versions of TclTutor had hardcoded lists of files containing lessons, which required modifying the main application whenever a new lesson was created. This got

old fast.

The lesson files are formatted as several sections with identifying strings to mark the start and end of a section. This

The next version of TclTutor used `grep` to find a keyword in the lesson files and built the menu on the fly. This worked until Tk (and TclTutor) was ported to Windows 95. Since then, there have been variation on the theme of reading in a lesson file, finding the keyword, and building the menus at run time.

The paradigm of data-driven, rather than code-driven GUIs is easy and commonin Tcl/Tk programs, but much less easy and much less common with GUI tools designed to work with compiled languages.

Things done Right and Wrong

Right

There are two design features that make using TclTutor significantly better than reading a book or using a pure HTML tutorial. These are the multiple levels of explanation and the ability to run, modify and rerun the examples.

The multiple levels of terseness allows an experienced programmer to view just syntax and example, while a less experienced programmer can get enough explanation to (hopefully) understand what the command does and why they might want to use it.

The ability to run, modify and rerun the examples is key to a learning experience. People learn better if they can apply information immediately after they acquire it.

The best implmentation decision I made was to create the course and lesson menus on the fly. This saved countless hours and innumerable bugs during phases lessons were in rapid development and frequent change.

Wrong

The biggest mistake I made was writing my own license for TclTutor. At that time the three models for releasing software were the original GPL, BSD or Shareware. I wanted TclTutor to be free to individuals, but not to allow anyone to sell copies.

I was afraid that someone would use my code and make money from it without giving me a share. In retrospect, this was unlikely. The TclTutor engine is relatively small and could easily be retro-engineered if anyone cared to.

The effect of my license was that as the Open Source movement started to become more aware of licenses, TclTutor was taken off Linux and BSD distributions that sold a distribution on a DVD/CD-Rom. Since most distributions are available on physical media for a small fee, this cut into the distribution of TclTutor just as Tcl was otherwise becoming more accepted.

The BSD license is better for small projects. The best kind of advertising a software engineer can get is to have developed an accepted product. The Open Source community is a great way to prove that you've "got the chops".

Another error was that rather than put TclTutor on a major website, I put it on my private site. The advantage of this was that I had direct access to the web logs and could count the number of downloads, see when it peaked, etc. The disadvantage is that not everyone in the world looks at my website to see what's new.

The worst implementation mistake was to use a keyword driven format for the lesson files. This is a classic example of writing a "C" application in Tcl. I could have saved myself a dozen or so lines of code and made the application more easily extended if I'd used a lesson format that looked like a Tcl script and loaded the lessons with the `source` command.

Future Work

There are always more lessons to be written. The Tk course has been in process for almost a dozen years. There are hundreds of packages that would profit by a TclTutor lesson or two.

The Release 3 version of TclTutor will use the BSD license, and will be available as a Starkit. I plan on making the application available on some website with wider distribution than www.msen.com/~clif.

And, there is always room for an improved GUI.