Nifty Assignments

Nick Parlante (moderator)
Stanford University
nick.parlante@cs.stanford.edu

John K. Estell
Ohio Northern University
j-estell@onu.edu

David Reed
Creighton University
davereed@creighton.edu

Jeff Lehman
Huntington College
jlehman@huntington.edu

David Matuszek
University of Pennsylvania
dave@acm.org

Donald Chinn
University of Washington, Tacoma
dchinn@u.washington.edu

Categories and Subject Descriptors
D.1.5 [**Programming Techniques**]: Object Oriented Programming. K.3.0 [**Computers and Education**]: General.

General Terms
Algorithms, Design, Languages.

Keywords
Education, assignments, homeworks, examples, repository, library, nifty, object oriented programming, pedagogy, pirate, cards, minesweeper, recursion.

Introduction
At SIGCSE, we mostly talk at a strategic level about broad ideas in CS education. Balanced against those important strategic discussions, we have the simple day-to-day need for quality assignments in the classroom, and that's where Nifty Assignments comes in.

Oftentimes, I like to think that what my students learn stems right from my explanations and examples in lecture. But one night of office hours watching them work through an assignment reminds me that most of the learning happens at those key, difficult-yet-inescapable points in the assignments. Lecture is left to play its supporting role of explanation and motivation, but in my heart, I suspect that lecture is theater compared to the learning that goes on when the students actually write the code.

Given the crucial role of assignments, I'm perpetually amazed at what an error prone and time-consuming process it is to put together a good assignment. The CSE community can be a great resource for this problem — sharing great assignment ideas and the materials to make them easy to adopt. That's what Nifty Assignments is all about.

Each presenter will introduce their assignment, give a quick demo, and describe its niche in the curriculum and its strengths and weaknesses. The presentations (and the descriptions below) merely introduce each assignment. For more detail, each assignment has its own web page, available from our home page **http://nifty.stanford.edu**, with assignment materials, handouts, data files, and whatnot. If you have an assignment that works well and would be of interest to the CSE community, please consider applying to present at Nifty Assignments. See the nifty.stanford.edu home page for more information.

David Reed — Talk Like a Pirate (CS0/CS1)
September 19 has been declared International Talk Like a Pirate Day, an (unofficial) holiday when people around the world are encouraged to say things like "Arrrr" and "Ahoy, matey" whenever possible. The Web site www.talklikeapirate.com contains a simple English-to-Pirate translator, which is entertaining to students and an inspiration for many nifty assignments.

In a recent Web-based CS0 course, the translator was used to motivate discussions on interface design, Web-based programming, and program extensibility. Initially, students critiqued the interface provided by the online translator, and designed a more general interface involving a separate button for each word or phrase to be translated. Students then implemented their own translators in the form of embedded JavaScript code in a Web page, and embellished the translator with special features (such as randomly inserting "Arrrrs").

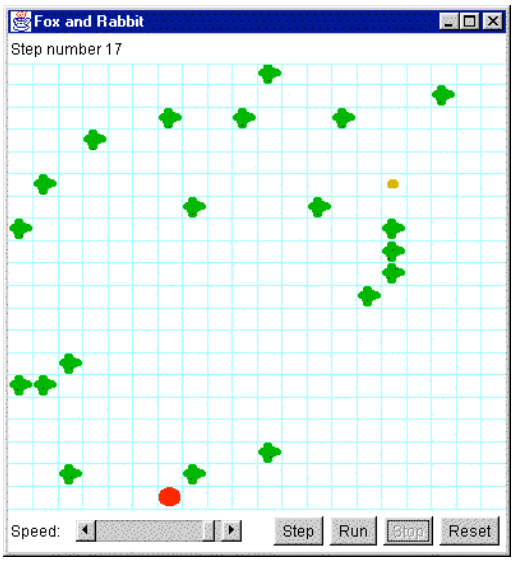
As students extended the vocabulary of their translators, they quickly recognized the amount of repetition involved. The next assignment involved using JavaScript to automate the generation of the buttons. By doing so, students experienced the value of code factoring and metaprogramming, resulting in a translator that was easier to extend, debug, and keep consistent.

While these assignments were given in a CS0 course using HTML and JavaScript, they could easily be adapted to a CS1 course using the Java GUI. In an applet/application, the user could enter text and have the program translate using pattern matching and string substitution. To be robust, regular expressions could be introduced to allow for context-sensitive searches (such as finding words that end with "ing").

David Matuszek — Rabbit Hunt (CS1)
A fox is hunting a rabbit in a field. The field contains a number of bushes which obstruct both the fox's view and the rabbit's view, so each may or may not be able to see the other. The fox tries to catch the rabbit; the rabbit tries to get away from the fox. If the fox can catch the rabbit, he eats it (and wins). If the rabbit can keep away from the fox for 100 turns, the rabbit wins.

The student is the rabbit. The student's score on the assignment is (with minor adjustments) the percentage of times that the rabbit wins, out of a large number of trials.

Copyright is held by the author/owner(s).
SIGCSE'04, March 3–7, 2004, Norfolk, Virginia, USA.
ACM 1-58113-798-2/04/0003.



As instructor, I provided the framework for the assignment (in Java, using AWT). The students had to replace one method, decideMove(), in the Rabbit class. In this method, the student could have the rabbit look in any of the eight compass directions, and note what it sees there (fox, bush, edge of field) and how far away it is. The Fox could do the same.

This assignment is unusual in a number of respects. It is very visual. There is no algorithm or "solution"—students have to figure out a strategy and implement it. To do well, students have to experiment with different strategies. There is a clear incentive to examine existing code (how the Fox decides its moves). Students have to decide for themselves when their strategy is "good enough."

I found this to be a very engaging assignment, and students who did well expressed a real sense of accomplishment.

Jeff Lehman – Minesweeper (CS1)

The game of Minesweeper requires a player to determine the location of "mines" located randomly throughout a two-dimensional grid or "minefield". The Minesweeper assignment grew out of a "never ending quest" to find application areas beyond the limited complexity and interest of "Fahrenheit to Celsius" problems. The assignment has evolved to demonstrate the value of an object-oriented versus a procedural approach and to illustrate the connection between a graphical user interface and a supporting class data structure. Variations of the assignment have been used in our CS1 course (more like CS1.5) using Java and in a post CS2 course focusing on Visual BASIC.

The "niftiness" of this assignment is that it can be used to cover and integrate a wide range of programming topics. The assignment can focus on procedural methods/functions incorporating two-dimensional array processing and recursion. Students implement and test methods/functions that support key elements of a Minesweeper game. The assignment can also focus on object-oriented design and programming. Students implement and test a Minesweeper class as a supporting data structure. The design for the class can be supplied by the instructor or be incorporated into the assignment. A set of classes can be provided so that students do not need to implement a GUI to test their classes. The assignment can be also used to introduce GUI design and programming where students create a complete user interface for their Minesweeper class. There are many opportunities for students to take the assignment "above and beyond" such as adding a timer, sound, and saving high scores.

John K. Estell — The Card Game Assignment (CS1-CS2)

Who hasn't written a card game? However, for most of us teaching programming courses, our code was written back in the days of command line input and text-only output. In contrast, our students have access to nifty graphical user interfaces and object-oriented languages such as Java that makes it easy to work with images — such as cards — in programs. Unfortunately, there are some obstacles: trying to obtain a set of card images that are not encumbered by any copyright restrictions, and guiding students toward a modular design approach featuring well-designed core card classes that are conducive for development of multiple card games.

A card game assignment is used in our third introductory programming course, where after two quarters of C++, students are exposed to GUIs, event handling, and code libraries (including the Collections Framework) using Java. A set of card images distributed through the GNU General Public License is made available for use. Students are instructed to write a test engine to verify the correctness of their card class prior to implementing the card game itself. It is a nifty assignment in several ways. Most students are familiar with card games, Java Swing components facilitate both displaying and interacting with images, and the program actually does something fun once it is completed. Additionally, the assignment presents an opportunity to discuss ethical issues such as copyrights and licensing. The accompanying web site provides both the set of card images and several classic, yet doable, card game assignments for your adopting pleasure.

Donald Chinn — Digital Signatures, Font Files, and Recursion (CS2)

Many examples of recursion either are too simple (for example, factorial or Fibonacci numbers, mergesort, quicksort), or could be solved directly and more efficiently by iteration (for example, searching through a binary search tree). Even recursion problems of appropriate complexity may suffer because the problem looks contrived, such as with the Towers of Hanoi. This assignment centers on a problem that combines significant recursion in a real-world problem using cryptography, security, and font files.

Under some circumstances, a font file is modified to include only those characters used in a document. This process, called subsetting, can drastically reduce the space used by a font. However, if we attach a digital signature to a font file to detect that the font has been modified from the original, then the signature on the original font file will no longer be valid if it is subsetted. Signing the file again after subsetting is not a feasible option.

This assignment asks students to implement a recursive solution to the problem of preserving the authentication power of a digital signature, even when arbitrary subsets of the original file are removed. It is an application of recursion that is of moderate to high complexity, not easily implemented by an iterative solution, and based on a real world application.

Successful completion of the assignment requires a knowledge of recursion and binary trees. Previous experience with recursive algorithms such as mergesort could be helpful. The assignment could be given in an algorithms course or as an advanced exercise in the data structures course. Its difficulty and length can be adjusted by varying the amount of starter code, examples, etc. provided to the students.