Nifty Assignments

Nick Parlante, Julie Zelenski
(moderators)
Stanford University
nick.parlante@cs.stanford.edu
zelenski@cs.stanford.edu

Stephen Davies
University of Mary Washington
stephen@umw.edu

Joshua T. Guerin
University of Kentucky
jtguer2@uky.edu

Debby Keen
University of Kentucky
keen@cs.uky.edu

Zachary Kurmas
Grand Valley State University
kurmasz@gvsu.edu

Dave O'Hallaron
Carnegie Mellon University
droh@cs.cmu.edu

Kevin Wayne
Princeton University
wayne@princeton.edu

Daniel Zingaro
University of Toronto
daniel.zingaro@utoronto.ca

Categories and Subject Descriptors
K.3.0 [**Computers and Education**]: General.

General Terms
Algorithms, Design, Languages.

Keywords
Education, assignments, homeworks, examples, repository, library, nifty, pedagogy.

Introduction
I fiddle with topics in the syllabus, edit demos and presentations and entertain myself with fun slide transitions. And yet, inevitably, I must accept that most of what my students learn and remember from my course comes from the assignments. Great assignments are hard to dream up and time-consuming to develop. With that in mind, the Nifty Assignments session is all about promoting and sharing the ideas and ready-to-use materials of successful assignments.

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. A key part of Nifty Assignments is the mundane but vital role of distributing the materials – handouts, data files, starter code – that make each assignment ready to adopt. The Nifty Assignments home page, **http://nifty.stanford.edu**, gathers all the assignments and makes them and their support materials freely available.

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.

Nick's standing ACM editorial: a large part of the value of Nifty Assignments is that all the materials are on the internet for anyone to see without screening people (or search engines!) out by some affiliation or login. In keeping with the ACM's mission to promote Computer Science, the ACM should raise funds in some other way, such as by charging authors. The ACM materials should be simply and freely available on the internet.

Exploring Loops with Sounds (CS1) - Daniel Zingaro
A typical early CS1 assignment has students use loops to solve small programming problems. Rather than relying on variations of string or array problems, this assignment has students loop through sound objects to perform interesting transformations on those sounds.

Students begin by writing a function to remove vocals from music. The technique is particularly simple, relying on a crude generalization of how music is recorded. Importantly, the function is a pure loop function, with no if-statements or booleans. Since public domain vocals are difficult to find, I provide students with wav files whose vocals can be removed with varying levels of success.

Following this warm-up, students write functions to fade-in and fade-out a sound, then compose these functions in order to add both types of fades in a single call. Finally, building on what was learned about fading, students write a function to make a sound move from left to right in the stereo field. Subtle but important differences between fading and panning lead to an understanding of the use of helper functions to avoid code duplication.

With a suitable abstraction of sounds and samples, each function can be written in around fifteen lines of code. The assignment gives students practice with loops, processing sequences, and function composition. Students should be encouraged to experiment with the vocal-removal algorithm. Students with interest in sound-editing may wish to create songs, adding vocals in various ways to see how well the algorithm can remove them.

Plucking a Guitar String (CS1) - Kevin Wayne
Simulate the plucking of a guitar string with the Karplus-Strong algorithm and transform your laptop into a musical instrument. The assignment illustrates a basic programming construct (a data type that stores a sequence of values), takes advantage of a fundamental computer science concept (digital audio represents sound as a sequence of amplitudes), and shows the role of computation in an important application (physically-modeled sound synthesis).

Karplus-Strong models the vibration of a guitar string with a remarkably simple process: Maintain a buffer of amplitudes; then, repeatedly delete the first amplitude from the front of the buffer and append the average of the first two amplitudes to the end of the buffer. The size of the buffer determines the frequency of the guitar string.

To represent a guitar, you will create many guitar strings (of different frequencies), process keystrokes to identify which strings get plucked, superpose the resulting sound waves, and sonify the results. The key data structure needed to perform the Karplus-String simulation efficiently is a ring buffer (bounded queue).

The assignment is designed for late use in a CS1 (or very early use in a CS2) curriculum. It relies on a real-time audio library with an extremely simple API — we have developed versions in both Java and Python. There are numerous opportunities for creativity, enrichment, and inspiration: perform a musical piece on your laptop, plot the sound wave as the user plays the keyboard guitar, or modify the Karplus-Strong update rule to design a new instrument

Tournament Uno (CS1) - Stephen Davies

One of the difficulties a CS1 instructor faces is the large gap in incoming skill sets. Some students are taking the course as a true introduction to programming, while others have been self-taught hackers for years. How can an instructor truly challenge the programming-savvy while at the same time not overwhelming novices with an overly complex assignment?

One way is by introducing good-spirited competition within the class. Uno! is a simple but popular card game known around the world. The complexity involved in simply following the rules is modest, but there are a surprising number of choices players unconsciously make as they play. (Should I match the up card's rank, or color? Should I play my wild card now or hold it? Does the person in the lead come immediately after me, and would it be wise to use a penalty card on them? What should I change the color to? etc.)

This nifty assignment comes equipped with a Java simulator program that can simulate thousands of consecutive Uno! games in seconds. Only one thing is missing: the pluggable strategy method for each player. Students formulate their strategies and write code to choose which card to play, which are then plugged into the simulator and compete against each other in a bracket-style tournament. It's surprising how motivating it is to even the most advanced students to try and reach the Final Four!

A PPM Image Editor (CS1-CS2) - Joshua T. Guerin and Debby Keen

Undergraduates often come into introductory-level computer science courses with expectations that match their past, multimedia-heavy computing experiences. Thus, programming assignments with graphical elements are often a favorite for intro-level CS courses. However, many commonly used programming environments don't facilitate easy use of computer graphics. Even in the case where simple graphics libraries are available, they are generally language (and platform) dependent and often favor a "black-box" style of coding that doesn't encourage a deep, low-level understanding of the underlying algorithms or data structures.

The PPM image format offers instructors a powerful tool for grounding graphics assignments in the real world. The PPM specification allows for ASCII representation of an uncompressed image, making image-editing possible in the language and platform of the instructor's choice, without the need for external libraries. Students can create seemingly advanced tools using simple (CS1 or CS2) level control and data structures. Students gain the experience of creating interesting image filters from scratch, as well as a deeper understanding of the underlying algorithms and representations used to manipulate images.

In our teaching resource package we have provided several example filters from which the instructor can choose, including pixel-level effects: color inversion, black and white / grayscale conversion, adjustment of brightness/contrast/noise, and image-level effects such as flipping and blurring.

Igel Ärgern (CS2) - Zachary Kurmas

Igel Ärgern is a simple and fun German board game whose name is commonly translated as "Hedgehogs in a Hurry". This board game makes for an excellent CS2 project because (1) it brings together many CS2 topics (two-dimensional arrays, stacks, inheritance, polymorphism, design, GUI, etc.), (2) it has an unusually large number of rule variations, and (3) the finished product is a real, published game that both adults and children enjoy playing.

The main idea of the game is that players race their four hedgehogs across the board; however, there are two challenges: (1) Hedgehogs can stack on top of each other; and only the top hedgehog in the stack may move. (2) There are obstacles (denoted by black squares) on the board. The rules suggest many different behaviors for these obstacles (concrete blocks, pits, etc.). Implementing different behaviors for the obstacles provides the students an interesting exercise in inheritance and polymorphism.

In addition to different obstacle behavior, there are many other rule variations that that can be used to either modify the project from semester to semester or to encourage student creativity.

The current version of this project is designed as a large, six-week semester project with no starter code; however, it can easily be scaled down into one or more smaller projects (e.g., by providing the GUI and having the students implement the game logic only).

Binary Bomb (CS3+) - Dave O'Hallaron

A *binary bomb* is a device that makes it fun for students to learn assembly language. The scenario is that the nefarious Dr Evil has planted a bunch of *bombs* on our system. We ask the students to help us defeat Dr. Evil by defusing one of these bombs.

A *bomb* is a C program that consists of *phases*. Each phase expects the student to type some string. Typing the correct string *defuses* the phase. Otherwise the bomb *explodes* by printing "Boom." In either case, the bomb sends the string to a server, which validates it, awarding points for each defused phase and deducting for each explosion. The server displays statistics for each bomb on a real time scoreboard. The twist is that students get only a binary executable. To defuse their bomb, they must reverse-engineer this binary, understanding the assembly code for each phase.

The bomb is beautiful in many ways. Although we don't expect intro courses to adopt this particular lab, it can teach us a lot about designing good assignments. The bomb is *fun*, turning a dry topic into a game. The bomb requires *reverse engineering*, a type of thinking that gives deep insight and forces students to learn tools like debuggers. The bomb is *polymorphic*. Each student gets a unique autogenerated bomb, which helps reduce cheating. The bomb is *interactive*. Students love to see their peers' progress on the real-time scoreboard. The bomb is *autograded*, scaling to classes of arbitrary size.