Audiometer Simulation Project Summary

Paul R. Potts

March 1993

In my first "real" job out of college (not counting a stint doing word processing for the Department of Anthropology), I worked for the University of Michigan's Office of Instructional Technology developing multimedia for teaching. The Audiometer Simulation project was a collaboration with Dr. Paul Green of the Department of Industrial and Operations Engineering. In 1993, as part of the process of archiving some older projects, I wrote the text below, answering a template of questions we asked all the project developers to use. I was not brief. Oh well.

Not much remains of the Audiometer Simulation project that can easily be demonstrated today, as it ran on old Macintosh II family computers and used HyperCard. It may be possible, though. I have previously used SheepShaver to run an emulated old MacOS environment on MacOS X on Intel, which allowed me to run old MacOS software including Microsoft Word and ResEdit, and access old files on a mounted disc image, but I have not looked into doing something similar on M-series Macs like my current MacBook Air M2. I have also previously had some success sending an old personal HyperCard stack to the Internet Archive and getting it to run on their online emulated HyperCard player, although it tends to crash and freeze up. I would not expect the emulation of the System 7 sound driver and Macintosh II sound hardware to work smoothly on either platform.

1. Describe the history of the project in brief.

The project began in December of 1990 and was the first development project that Paul Potts was involved in. Experimental prototypes were created during the first few months. The early attempts involved trying to work with short clips of digitized sine waves, but these could not be played in a loop from within Hypercard without producing a repeated clicking sound.

Paul Potts decided to write an external command in THINK C which would provide generalized support for sound without Hypercard's overhead. This was done in the form of a small command interpreter and set of sound object classes designed to encapsulate the tricky Sound Manager code and provide good error-checking. This involved several new techniques, including: writing a "persistent" XCMD that remained in the heap between calls; using the ANSI-A4 library to support standard library functions like sprintf() and sscanf() within a code resource; using THINK C's object-oriented extensions; and writing a small tokenizer.

At the time System 7 beta drafts were becoming available, and documentation on the Sound Manager in technical notes, Inside Macintosh, and the drafts of Inside Macintosh volume VII was incorrect and contained a number of errors. This led to delays and difficulties. Eventually technical support came from Jim Reekes, the Apple programmer who designed the Sound Manager. An early version of the code that was used for quite some time was very unstable because of a bug which caused it to allocate a sound channel in low memory. The Sound Manager does not provide any error-checking for mistakes such as this. The code would function for two or three times and then the Macintosh would crash spectacularly with lots of noise and garbled video. This was eventually traced to a single failure to correctly initialize a C pointer.

Various other technical challenges were overcome and bugs were worked out of the code and the deployment systems. Here is an example of a bug: on Dr. Green's Macintosh with a Radius two-page display, the audiometer window would be drawn but sized too small, and it could not be resized to its full size. This turned out to be a conflict between Hypercard and the system extension to provide support for the full-screen display. Memory was allocated for the window when the application started up, and then the Radius extension placed a non-relocatable block on the application's heap, which would not allow the memory block to grow when the window was resized. Hypercard would not deallocate and reallocate the window record. This was eventually solved by installing a later version of the Radius extension which eliminated the problem.

The code to provide the logic for the audiometer was written in the Hypertalk scripting language. A second stack which ran in another window was used to graph the user's performance. The first working version of the Hypertalk code was not well-tested and was extremely slow, but functional. Most of the difficulty in the Hypertalk code was in the code that teaches the student the proper technique for administering a hearing test. The scripts that simply allow the XCMD to work like a standard audiometer are relatively simple. Analyzing every possible move the student can make for legality and providing appropriate instructions at every step proved much more difficult.

Draft versions of the XCMD code were distributed via the Internet to various parties interested in debugging and testing, including to Jim Reekes, Apple Sound Manager author. Reekes made a number of suggestions for improvements to the sound code. Most of these were not implemented because they were not necessary for the project, but one, making the sound play asynchronously, was implemented for better compatibility with other applications and future versions of the system software.

A stable version of the XCMD was ready for testing in October 1991. Students had the opportunity to use the software in class. The pilot deployment was considered to be a useful learning experience but not a useful teaching experience.

A number of problems cropped up with the scripts. Students considered the execution of the program to be much too slow. The machines provided had only 1 megabyte of RAM. At the last minute a second megabyte was installed, but running a large Hypercard stack in two megabytes could cause problems. Speed was very poor in part due to the fact that when Hypercard runs low on memory, it jettisons compiled Hypertalk code, and then must recompile it "on the fly" when it is called again. This can result in long pauses at run-time.

At the end of 1991, System 7 had just been released. Paul Potts spent some time learning about AppleEvents and conducted an experiment to demonstrate the feasibility of controlling another program remotely from a Hypercard stack. He placed the resuable audiometer code module into an application shell written using THINK C, and ran six copies on a networked lab of Macintoshes, then used a single Macintosh running a Hypercard stack to control sound play on the remote Macintoshes. The performance was much too slow to use for the audiometer project, but AppleEvents may play an important part in future OIT projects.

Another source of difficulty with the hardware was in the cabling and calibration process. The cables provided were extremely touchy and would occasionally stop working in the middle of a hearing test. Also, it was determined that calibrating the audiometer could not be achieved with electronic devices such as voltmeters because of the very low voltage levels produced by extremely quiet sounds. These had to be estimated by ear.

In the summer of 1992 the audiometer program was rewritten using all new logic as a functional heirarchy and state machine, broken into short script modules. This has been a great aid in locating bugs in the script code. Many of the problems with the script code would not have occurred with another language such as C or Pascal. Hypertalk does not provide language constructs for supporting abstract data types, data encapsulation, separate compilation, enumerated types, and other features. Many of these were "simulated" in Hypertalk to improve the design and efficiency of the program. In general these simulated features worked well, but it proved difficult to properly debug the individual scripts, even though they were broken into small parts, due to the complex interactions between scripts that stem from a lack of proper data encapsulation. Many scripts were tested and optimized to improve the speed.

In October of 1992 the program was used again with several classes. This time sufficient memory and computing horsepower was available, although there were still some difficulties with the cables. The software performed well. Only one problem was found, and quickly corrected during the class. The speed was improved, although still somewhat sluggish. It became clear after class use that the biggest bottleneck was the process of switching between the graphing stack and audiometer stack.

In November and the first three days of December the stack was altered to support a larger window, and the graphing was incorporated directly onto the page with the audiometer simulation. This improved speed significantly. A small change to the logic requested by Paul Green resulted in a number of new bugs and pointed out to me the difficulty of attempting to use techniques that I use for traditional software without language constructs to adequately support them. Some changes in messages, card layout, and user interface were also introduced.

A completed and revised version, 1.2, was finished on December 3rd.

2. How has the project been used in an instructional setting?

In October of 1991 and 1992, the stack was provided running on two computers alongside two traditional audiometers. Students were required to use the audiometers both as subjects and as testers.

Students found the simulated audiometer difficult to use, slow, and somewhat unstable. They were interested in the concept of the project and did manage to learn the basic technique of conducting a hearing test, but it was clear that more work needed to be done before the computerized audiometer could provide true enrichment of the learning experience. The student comments reflected both their interest in the technological approach and their frustrations with it.

In the following year, the audiometer software performed much better. There were a few difficulties with the cables, but for the most part the student experience was quite positive. The evaluations returned by students were almost uniformly positive.

3. What is the status of the project vis-a-vis the current agreement for joint development?

The audiometer in its current stage has met the objectives of the agreement for joint development, and both parties (Paul Green and Paul Potts) are reasonably satisfied with the software.

4. Are there remaining bugs or known problems with the software? If so, describe.

There are no known bugs with the current version of the software. However, the software is still somewhat difficult to use properly because of the necessity for custom-built hardware (the audio cables) and calibration by ear. The accuracy of the tones could use improvement, although I don't believe that any improvement is possible using current Macintosh sound output.

5. Describe any features which were left out of the software or which could be added if it was developed further.

If this is developed further, it would be helpful to translate the script code into a high-level language such as C++ or THINK C with the class library. Using a traditional language such as C would allow for true abstraction of the data types, improved debugging facilities, and improved speed. I believe that by the time the project was rewritten and the script code debugged, the advantages in speed and simplicity gained in writing the script code in Hypertalk had been lost. Using C or C++ with a good library for the user interface would now be an effective approach.

6. If evaluation was performed, what was the overall result?

Evaluation was performed in October of 1991 and October of 1992. The software satisfaction survey form was used. These completed forms are included with the project archive. The general trend was that in the first use, users were interested in the project but very frustrated with the speed problem and errors. The second batch of evaluations were uniformly positive and rated the software as beneficial to the learning experience.
