



MTO 3.5 Examples: Brinkman and Marvin, Using the Tools to Teach the Tools

(Note: audio, video, and other interactive examples are only available online)
http://www.mtosmt.org/issues/mto.97.3.5/mto.97.3.5.brinkman_marvin.php

Figure 1. Course Syllabus

TH 423

**Computing for Pedagogical
and Cognitive Research Applications**

Spring, 1997

Course Description:

This course develops HyperCard programming and multimedia skills for application in music theory pedagogy and in music-cognitive research. We will present HyperCard authoring tools and HyperTalk scripting, and will introduce external commands and commercial extensions for integration of MIDI sound, sound files from commercial CDs, music notation, and scanned images. Students will develop criteria for evaluating existing CAI and apply these to selected programs, and will learn pedagogical principles behind effective stack design. In addition, the course provides an introduction to music-cognitive research, including basic concepts in experimental design and data analysis. Students will critique published experiments, design and carry out a group experiment implemented in HyperCard, and summarize the experiment's context and design in writing. Final individual projects will combine the skills developed throughout the semester by means of a HyperCard stack for computer-assisted instruction.

General Syllabus:

Weeks 1-3: HyperCard and HyperTalk Basics

Weeks 4-6: Importing Sound and Visuals; Introduction to Cognitive Research

Week 7: Stack Design Principles for CAI; Advanced Audio Toolkit

[Semester Break]

Weeks 8-9: Experimental Design; More on Scripting with HyperTalk

Weeks 10-12: Detailed Design Principles for CAI; Experimental Trials

[Jury Week]

Weeks 13-14: Evaluate Experimental Data; Finish Final Projects

Required Texts:

George Beckman, *HyperCard 2.3 in a Hurry* (Wadsworth, 1995).

Danny Goodman, *The Complete HyperCard 2.2 Handbook, 4th Ed.* (Random House, 1993).

Recommended Texts:

David Butler, *The Musician's Guide to Perception and Cognition* (Schirmer Books, 1992).

Instructors:

Aleck Brinkman, Annex M802; 274-1553 (342-3922); email: aleck@theory.esm.rochester.edu

Betsy Marvin, Annex 410; 274-1551 (328-2464); email: betsy@theory.esm.rochester.edu

"Office Hours" in the lab after class and by appointment (feel free to call us at home to schedule)

Course Requirements:

DAILY ASSIGNMENTS - Students are expected to prepare for each class, either by reading assignments and/or designing and implementing stacks. Each assignment builds on the previous one, so it is essential that students not fall behind.

ON-LINE JOURNAL - Students will be expected to examine and evaluate three existing software packages and to keep a journal, using an on-line HyperCard template. Despite the informal name "Journal," we expect polished writing and academic integrity, i.e., do not "cut and paste" from software or documentation unless you quote the material and give proper citations. Students will prepare a one-page summary of one assigned program to share with the rest of the class (make 16 copies).

CLASS PROJECT - Students will engage in music-cognitive research by means of a jointly designed experiment implemented using HyperCard and external commands. Although the project will be developed as a group, students will turn in individual papers in APA style that report on the experiment's context and method.

INDIVIDUAL PROJECT - The final project for the course will be a HyperCard stack demonstrating mastery of techniques developed over the course of the semester. Topics will vary according to the interests of the individual.

TIMETABLE:

March 7 -- Journals due (before Spring Break)

March 10-14 -- Work on final project design ideas (Spring Break)

March 24 -- Final Project Title Page due (must include menu showing overall design of project)

April 21-25 -- Individual appointments for project progress check (Jury Week)

April 28 -- Psych papers due (Introduction and Method for Experiment)

May 14-16 -- Presentation time TBA during exam period

Evaluation/Grading:

Daily Assignments 25%

On-line journal 20%

Psych paper (Intro/Methods) 15%

Individual project stack 40%

Figure 4. Evaluation of Software (Class Outline)

TH 423 -EVALUATING COMMERCIAL SOFTWARE

Ideas for on-line journal stack

ITEM OVERVIEW:

- Name of program
- Author
- Publisher
- Hardware requirements
- Support phone number
- Publication Date
- Audience
- Objectives
- Published Reviews
- Overall rating

USER INTERFACE:

- Visual Style, such as:
 - fonts (readily available, legible)
 - integrated modalities (sound, graphics, speech, etc.)
 - consistent interface from card to card (intuitive)
 - layout (clutter-factor, icons instead of words, etc.)
 - attractiveness
 - consistent placement of navigation buttons, etc.
- Navigation
 - Clear indication of where you are and where can go
 - Consistency of buttons and fields (location style)
 - Standard buttons (home, help, next, previous, etc.)
 - Path through material (linear vs. nonlinear)
- Instructions
- User friendly?
 - Can you get to it from any card
- Mode (menu, pop up)
- Clarity
- Concision
- Optional?
- Sample run
 - Mode of user input (typing, midi keyboard, check box, can user take notes on-line? undo?)

GENERAL DISCUSSION -- Content Depends upon Type:

1) Tutorial (content oriented)

Consider the following:

- Content & Sequence (useful information?)
- Are good examples given?

- Is there some student involvement?
- Does it test mastery? Immediate or delayed feedback?
- Does it allow different pacing?

2) Drill and Practice

Consider the following:

- Content & Sequence
- Pedagogical Appropriateness
(does it develop the skills it is supposed to?)
- Appropriateness of medium
- Is the activity fun, engaging?
- Is speed a factor?
- Feedback
- immediate or delayed?
- branching for remediation and advancement?
- automated record keeping?
- use of humor, sound, graphics?
- mastery levels to be attained before advancing?
- adequate randomization of drill items
- can student specify subcategories? (e.g., 3rds and 6ths only)

3) Games

Consider the following:

- Content & Sequence
- Speed a factor? (Beat the clock, etc.)
- Number of players (networking)
- Motivation issues (fantasy elements? something to add fun)
- Branching: does it get harder as you go?
- Pedagogical appropriateness: does it exercise useful skills?
- Feedback: use of humor, sound, graphics?

4) Simulations

Consider the following:

- What is simulated and how?
- Why use a simulation, is it appropriate?
- Content & Sequence
- Are good examples given?
- Is there some student involvement?
- Does it test mastery? Immediate or delayed feedback?

CONTENT:

- more detail

OTHER:

- anything else?

Figure 9. Scripting Assignment 1

Preliminary:

First, Create a new stack, get the Message box, and set the userlevel to 5. Make a card button called "SetUp", place it in the center of the card, and open its script (command-option click on the button). The object of the first task is to write a button script for this button that will make a new field, set its properties; then make a button and set its properties. Until you are comfortable with the syntax of all of the commands, the best way to do this is to test each command in the message box, and then, when it works properly, use cut and paste to copy the command into the button script. For example, for a., below, type

```
doMenu "new field"
```

in the message box. If the command works properly, copy the contents of the message box (select the text and then command-c), and paste it into the button script (place the cursor, and then command-v). If the command does not do what you want it to, experiment in the message box until you get it right, then paste it into the button script. Note, throughout this assignment, if you ever have trouble accessing the menus you need, reset the userlevel to 5 (especially if you started in one session, and continued in another session).

Instructions how to word each of the commands needed to complete this assignment are found in the Goodman text. Use this as an opportunity to become familiar with this resource. Numbers in square brackets refer to page numbers in Goodman The Complete HyperCard 2.2 (4th ed.).

Task 1.

- a. Create a field (use doMenu, and be sure to put quotes around the menu item in your command and use the exact wording from the menu). [See pp. 602-605.]
- b. Set the name of the field to "Title" [pp. 573-574]. Remember, you are doing this from the message box, not with the field tool, and that you can refer to the field you just created as "the last card field." (Throughout the assignment, in your scripts or message box, be sure to specify card field, or HyperCard will default to background field and tell you it can't find background field "title")
- c. Set the size (rect) of the field [pp. 684-685] to about half the size of the card (hint: you can get the coordinates of the card by typing "the rect of card 1" in message, then experiment with the rect of the field to choose a size you like
- d. Position the field in the middle of the card (figure this out from the rect of the card)
- e. Put the title "Scripting Assignment No. 1" into the first line of the card field [pp. 474-477].
- f. Put your name in the 4th line of the field.
- g. Set the font for the field to "Times," the style to "bold" and the fontsize to 23 [see handout on field properties and the appropriate pages in Goodman].

- h. Set the properties of the field so that it is opaque, and the lines don't show, and the text is locked.
- i. Make the font smaller for the line with your name in it, but not for the rest.
- j. Now use doMenu to make a new button. (Note, you can refer to this button as "last button".) Rename it by typing in message box:

```
set the name of last cd btn to "Next"
```

- k. Set the size of the button to match the size of card (use 'the rect')
- l. Hide the name of the button, and make it transparent (set the showName and the style of the button to appropriate values). [See handout on buttons and appropriate pages in Goodman.]
- m. Put the following into the script of the card button "Next":

```
on mouseUp
    go to next card
end mouseUp
```

As usual, try it first in Message but this eventually will go within your "Set-Up" button script. See the scripts 'Ralph', etc. in our handout illustrating field properties for an example of how to do this. So your command will begin:

```
set the script of cd btn "Next" to "on mouseUp" & return ...
```

This will be longer than the message box is, so type carefully. Peek at the script of the card button to see if your message box command worked properly. Once it does, we'll paste this into the "Set-Up" button script. To copy this whole command (since it is too long to be seen in Message), put the cursor in Message and hit cmd-A to select all, then cmd-C to copy it. Then go back and paste it into the "Set-Up" button script. Once you do this, you will need to add soft returns (option-return). The ampersand (&) is the concatenation operator and 'return' is a carriage return [pp. 821-823 and 830-831].

- n. Make a new card (doMenu). Now go back to the first card, and using the regular button and field tools, delete all buttons and fields except the "Set Up" button. Then click on "Set Up" to make sure that everything works.

You will note that the last button is still "selected." You can fix this by adding the the following command to the script for the Set Up button, after the Next button has been created:

```
choose browse tool
```

[See Goodman, pp. 504-506.] So the user doesn't have to see all the steps involved here, you should add a line at the beginning of your set up button script to lock the screen [pp. 553-555] before HyperCard creates the field, and unlock it at the end after all steps for making the field and button are complete. (See scripts "Ralph," etc. from the previous class.) In order to test your script, you may have to delete the field and transparent button using the field and button tools (or message box).

Let's add a card script to do all the deletions for us:

```
on closeCard
    delete card fld "Title"
    delete card btn "Next"
end closeCard
```

To get to the card script you have two choices: cmd-option-c (which may not work on the library machines) or go to

the Objects menu, choose Card Info, and click on "Script" inside its dialog box.

Let's get some practice with a stack script as well. Write a stack script that sets the userlevel to 5 on openStack. (You open the stack script with cmd-option-S.) This is necessary because the New Button and New Field commands are not available unless you are at level 5, and we want this stack to work when the user opens it without having to set the userlevel.

Task 2:

This task teaches you to get the users name, store it in a global variable, and use it in various commands. You may use the appropriate menu tools as needed to do this task, and refer to Goodman on the "ask" and "answer" commands [pp. 549-553]. Make two "handlers" in the script of the second card. The first (on openCard) should use the "ask" command to get your name; and store it in a global variable called "name". Hint: the ask command puts the user's input into the local variable It [pp. 402, 466]. You'll need a command to

```
put It into name
```

Do not call it "global name". When you use this global variable, each handler that uses it must have the line:

```
global name
```

Remember to end the handler properly.

The second handler (on closeCard) should use the "answer" command to say "goodbye" to you, using your name, and provide an appropriate response button (something, like "See you later" or "Bye, now"). Hints: (a) you will need to declare the same global variable as in the first script; (b) the '&&' operator concatenates (joins) two strings, like '&', but adds a space between them [pp. 821-823]. Now make a button (using the regular menu tools) to take you to the next card; choosing an appropriate icon. Write the script yourself, rather than using authoring tools. Test the button and the scripts. You might want to use the drawing tools to put something on the card, it's kind of dull with nothing there!

Task 3:

Make a third card. Using the regular button tool, make a "Create Field" button that contains a script that makes a new field, names the field, sets its properties, and places several lines of text in it. [The button script will do many of the things you did in Task 1.] Again, using the regular button tools, create a "Delete Field" button that deletes the field. Add several other buttons (with the button tool) that change properties of the field or its text, and that change properties of buttons on the card. [See handout on Field Properties and the appropriate Goodman pages.] You will need to name all of your buttons in order to do the latter. Next to each button, make an "undo" button, puts things back as they were. Experiment with changing properties of specific words or lines in the field. If your button only changes the properties of a portion of your field's text, you must specify which portion to undo [pp. 469-470], e.g.:

```
set word 2 of line 1 of card fld "text" to plain
```

Put appropriate navigation buttons on the card for going to the previous and next card. Try to make something interesting out of this, but the primary purpose is to learn field and button properties and to practice writing scripts.

Task 4:

Now let's name this handler and put it into the card script; we'll call it from the "Create Field" button. Copy card 3 (using the Edit Menu) and paste it into the stack as card 4. [It will look exactly like card 3; don't worry about that.] Open the button script in your new card 4 for the "Create Field" button, select the whole script (command-A), and copy it (command-C). Then replace the card 4's button's script with one that reads:


```
on mouseUp
    setField
end mouseUp
```

Now open the card script (cmd-option-C) and paste the handler you just copied from button "Create Field" into the card script (command-v). Change "on mouseUp" to "on setField" (remember to do this for the end handler as well.) Now go back and try the "Create Field" button to be sure it calls the new handler correctly. If you are feeling adventurous, you might replace other button scripts on this card with handlers (in the card script) as well. Be sure that each handler has a different name.

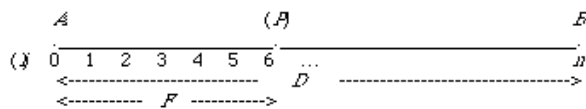
Fun With Numbers

We are going to create a stack that uses arithmetic calculations within repeat loops to move buttons around the screen. First, we will give you background information, then we will write scripts that move buttons in one direction only (horizontally or vertically), and then combine this information to move along a straight line between any two points.

READ: Goodman, Chapter 44 (Operators): carefully read pp. 809814 (arith. operators), scan p. 817825 for general familiarity, and read pp. 825826 (precedence).

Background

Suppose we want to calculate n equally spaced "steps" along a line from point A to point B.



1. The distance D from point A to point B is calculated by subtracting A from B. For our purposes, we will be measuring D in pixels. (Remember, the standard card width is 512 pixels and its height is 342 pixels. These are often given as x, y coordinates; the upper left-hand corner of the screen is 0,0 and the bottom right-hand corner is 512, 342.) We want to move along this line in n "steps."

2. If there are n points along this line, the fraction F of the distance D at any point i , is $(i / n) * D$. Thus when i is 0, F is $0/n * D (= 0)$; when $i = n$, F is $(n / n) * D (= 1 * D = D)$. At any arbitrary point i , the distance traversed is $(i / n) * D$. As a concrete example, imagine that point A is 0 and point B is 100; thus the distance (D) from A to B is 100 pixels, and that we want 10 steps (n) along this line. The step number, i , will vary from 0 to 10. At the beginning, when $i = 0$, the portion of the distance traversed is $(0/10) * 100 (= 0)$; after the first step it is $1/10 * 100 (= 10)$; after the second step it is $2/10 * 100 (= 20)$. After 10 steps whole distance has been traversed: $10/10 * 100 (= 100)$.

3. Since the starting point A usually isn't 0, the actual point P (where P is between A and B) is the partial distance F (which we calculated above) added to starting point A.

Let's take a simple example: Point A is 10, B is 30, and the number of steps n is 10:



The distance (D) from 10 to 30 equals 20. At the beginning of the line ($i = 0$) we are at point 10 $[(0/10) * 20 + 10]$. At point 1 we are at 12 $[(1/10) * 20 + 10]$; at point 5 (half way there) we are at point 20 $[(5/10) * 20 + 10]$, and so on. The last point (point 10) is $(10/10) * 20 + 10 = 30$. Plug some numbers into these equations and satisfy yourself that this is true, even if point A is greater than point B, i.e., we are going in a negative direction.

Task 0 -- Warm-Up; Trying It Out

We will now write a handler to test the simple math described above. Create a new stack, and set the userlevel to 5. Place the following handler in the background script of the stack (remember that you get the background script via option-command-B):

```
----- calc -----
```

```

-- Calc calculates n points on the straight line between points --
-- a and b. Note that round(<expression>) rounds <expression> to --
-- the nearest integer (whole number value). The values are      --
-- written into the message box so we can see them.              --
-----
on calc a, b, n
    put a into message
    put b - a into difference
    repeat with i = 1 to n
        put round((i / n) * difference + a) into newvalue
        put " " & newvalue after message
    end repeat
end calc

```

Now create a button called "Try Calc" and put the following into the button script:

```

----- Script for button "Try Calc" -----
-- Asks for two points and the number of steps, then passes these --
-- values to the calc handler                                     --
-----
on mouseUp
    ask "Point A" with "10"
    put it into startpoint
    ask "Point B" with "30"
    put it into endpoint
    ask "How many steps?" with "10"
    put it into steps
    calc startpoint, endpoint, steps
end mouseUp


```

Test the button and the calc handler by clicking on the button, entering numbers, and examining the results in the message box.

Hint: The font characteristics of the message box can be set just like a field or button. If you want to see more information in the message box, enter the command "set the textsize of message to 9" in the message box.

Task 1 - Moving Sideways

Now we are going to use our new skills to create "button animation." First use the button tool to make a new button called "x". Make it fairly small and roughly square, set its style to shadow, and disable it. We are now going to write scripts to make the button move. The handler moveh will move the button horizontally (from left to right or vice versa).

 \longrightarrow *F*

Recall that locations in HyperCard are given as comma separated values, e.g., '20,30', and that item 1 is the first value and item 2 is the second value. Add the following handler to the background script, replacing <expression> with the appropriate arithmetic expression to calculate the new value, based on the arithmetic we reviewed on page 1:

```

----- moveh -----

```

```

-- This handler moves the a button or field (obj) horizontally --
-- to point "newloc", using "n" steps. The second part of --
-- "newloc", the y co-ordinate, is ignored. Note that a point is --
-- expressed as (x,y); item 1 is the x value, and item 2 is the --
-- y value. Replace <expression> with the correct arithmetic --
-- expression --
-----
on moveh obj, newloc, n
  put item 1 of loc of obj into firstx
  put item 2 of loc of obj into yvalue
  put item 1 of newloc into lastx
  put lastx - firstx into xdifference
  repeat with i = 1 to n
    put <expression> into xvalue
    set the loc of obj to xvalue,yvalue
  end repeat
end moveh

```

Test this handler in the message box by entering lines such as

```
moveh "cd btn x", "256,200", 10
```

in the message box. (For now, type just as shown above-don't put quotes around "x"). Note that the first two arguments to moveh must be in quotes. For the command shown just above, the moveh handler will use "cd btn x" as the value of obj, "256,200" as the value of newloc, and 10 as the value of n.

Now, make a new button named "horizontal" and put the following handler into its button script:

```

on mouseUp
  ask "New Location" with the loc of cd btn "x"
  put it into newloc
  ask "How many steps" with "10"
  put it into steps
  put "card button" && quote & "x" & quote into object
  moveh object, newloc, steps
end mouseUp

```

When you click on the "horizontal" button, this handler will ask you for a new location and then move card button "x" to this location. Note that the handler uses the current location of the card button as the "default" value. Now test the "Horizontal" button and the "moveh" handler to be sure that they work properly. Note: if you move the button off of the card, just click on "horizontal" again, and enter more reasonable values for the new location.

Task 2 -- Moving Up and Down

In the background script, make a copy of the moveh handler (select the text, then command-c to copy and command-v to paste). Name the new handler "movev" (for move vertical). Remember to use change the last statement to "end movev" as well. Modify the handler to make it move the button in the vertical direction using the second (y) values and ignoring the first value (x).



Test the handler in the message box. After you get it working correctly, copy the "horizontal" button, rename it "vertical" and modify the button script of the new button so that it calls "movev" instead of "moveh."

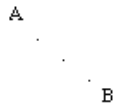
Task 3 - Going "around the corner"

Make another copy of the "Horizontal" button, name it "MoveX". The script of this button should call both moveh and movev to place button "x" in a new location using a "city-block" route, i.e., getting to the new location by going around the corner.



Task 4 - The Direct Route

Write a final version of the handler (on move ... end move) in the background script, that moves an object in a straight line to its new location.



This handler will combine elements of moveh and movev, i.e., for each point i, it will calculate a new xvalue and a new yvalue, and then set the loc of the button to (xvalue,yvalue). Test the handler in the message box, then make a card button that calls this handler to move card button "x". Note that since we defined the handlers with parameter "object" instead of using card button "x", you could use it to move any button or field to a new location. Experiment with this if you have time. Since the scripts are in the background, you can call them from other cards if you need more room to experiment.

Extra Credit

If you have had lots of previous experience or you found the above easy, feel free to design another task of your own that uses the techniques learned in this assignment.

Figure 17. Cognition Experiment: User Interface 3

Instructions	QUESTIONS	READY
<p>1. What is the meter of this excerpt?</p> <p style="text-align: center;"> <input type="checkbox"/> triple <input type="checkbox"/> quadruple </p> <p>2. Does the excerpt begin with an upbeat?</p> <p style="text-align: center;"> <input type="checkbox"/> upbeat <input type="checkbox"/> downbeat </p> <p>3. Based on the excerpt's style, does the overall harmonic structure conform to your expectations--that is, does it sound "right"?</p> <p style="text-align: center;"> <input type="checkbox"/> right <input type="checkbox"/> wrong </p> <p>4. Does the piece use imitation?</p> <p style="text-align: center;"> <input type="checkbox"/> non-imitative <input type="checkbox"/> imitative </p> <p>5. Is the key at the end of the excerpt the same or different from the first key?</p> <p style="text-align: center;"> <input type="checkbox"/> same <input type="checkbox"/> different </p> <p>6. Does the piece sound like it ends in the tonic key?</p> <p style="text-align: center;"> <input type="checkbox"/> non-tonic <input type="checkbox"/> tonic </p> <p>7. Is this piece familiar to you from having performed or analyzed it?</p> <p style="text-align: center;"> <input type="checkbox"/> unfamiliar <input type="checkbox"/> familiar </p>		

Figure 18. A MIDIplay Script

```

----- Play Midi Note -----
-- These button scripts cause a random tone to be played on a MIDI --
-- synthesizer, using MIDIplay external commands. --
-----

----- mouseUp handler -----
-- Sets timbre to piano or violin tone (50% chance of each). Generates --
-- midi note number randomly, and plays note for 1 second. Note: midi --
-- note numbers are integers, with 60 = middle C. Here we choose a --
-- random number between 1 and 35, and add 47 to it, so we get a note --
-- between 48 (C3) and 84 (C6). --
-----

on mouseUp
  if random(2) = 1
    then MIDIplay "xmit", "t 192 0" -- piano tone
    else MIDIplay "xmit", "t 192 40" -- violin tone
    put random(37) + 47 into note -- get random number between 48 & 84
    mplay note, 60 -- play note for 60 ticks (1 sec)
  end mouseUp

----- mplay -----
-- Sends a "note on" signal, waits x ticks, and sends "note off". --
-----

on mplay note, x -- note is midi note code (C4=60)
  MIDIplay "xmit", "t 144" && note && "100" -- note on
  wait x ticks
  midiplay "xmit", "t 128" && note && "0" -- note off
end mplay

```

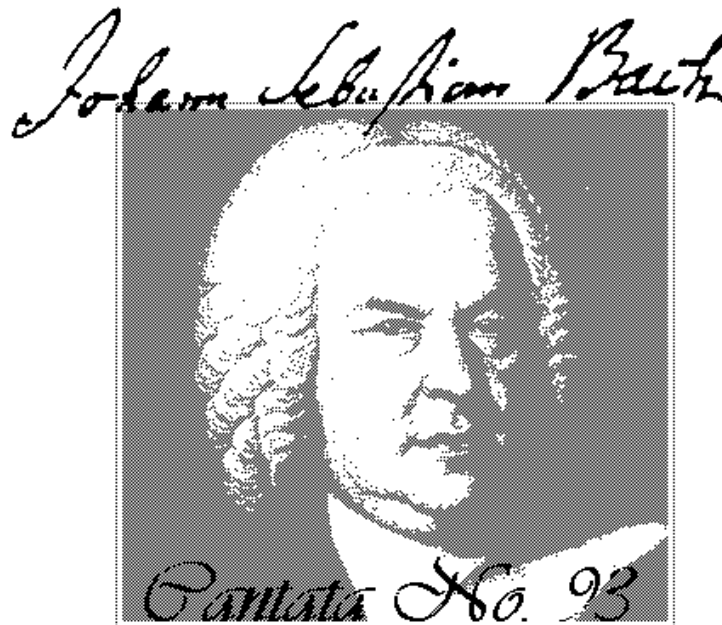
Figure 19. Final Project Guidelines

- There will be no specified minimum number of cards (nor a maximum number). Use whatever number of cards is necessary to present your material effectively.
- Your stack must demonstrate your mastery of some type of HyperCard-implemented sound, but you may choose among the types we have studied (internal Mac sound, MIDI files using MIDIplay, recorded sounds from CD using AudioShop or SoundEdit 16, or calls to CD using the Voyager CD Audio Toolkit), depending upon the nature of your project.
- If a large project is envisioned (one that is beyond the scope of the course), then demonstrate your vision of the entire project by setting up the "super-structure" of the complete project on your Table of Contents card. Clicking on lines in the TOC should take the user to various portions of your project stack, some of which will be fully implemented and others of which should simply contain a statement telling the user that this portion of the stack is not yet fully implemented. This statement should also contain a short prose description of what would be there in the completed project you envision.
- Above all, demonstrate your highest abilities on the portion(s) of your project that you choose to implement. Your grade will be based both on your conception and planning of the "whole," and the realization of the portions you have chosen to complete.

Figure 20. Dorhout Title Page and Main Menu

a. Title Page

The title page utilizes a scanned portrait of Bach, with his signature, and an attractive script font for the stack title.



b. Main Menu

The main menu features the various sections of the project: History of BWV 93, Text Translation, View the Score, The

Chorale, J.S. Bach's Life, The Performance, Review Quiz, plus Help and Quit. Clicking on each menu item takes the user to a submenu for that portion of the stack. The design is consistent, with a CD Controller in the lower left corner of each card, and a miniature portrait of Bach in the lower right. Clicking on the latter returns the user to the main menu.

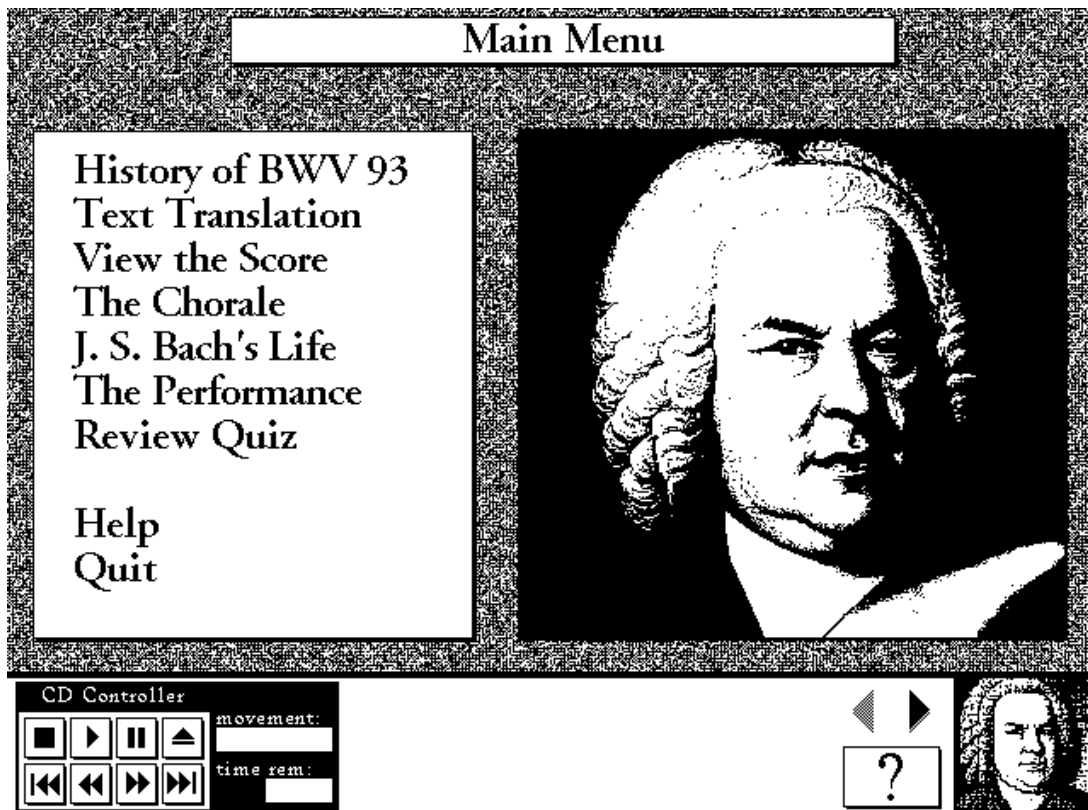


Figure 25. Fowler's Stack

a. Title Page

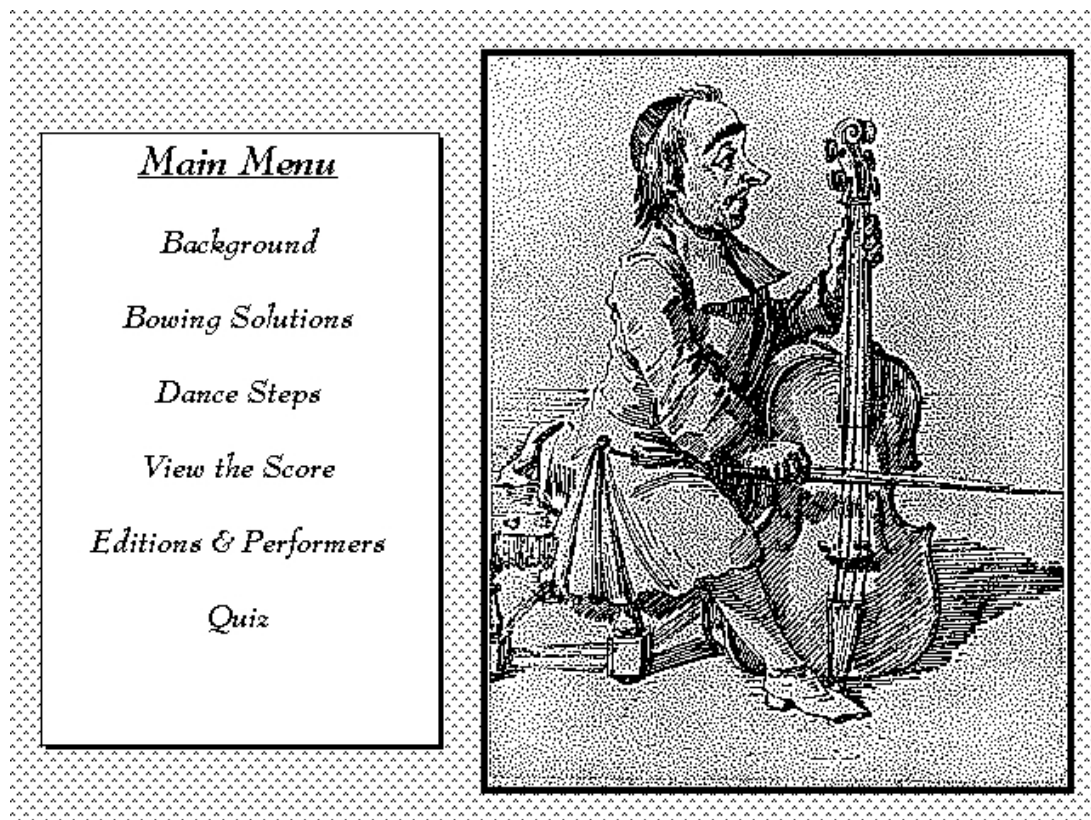
Shows the title of the project, "J.S. Bach Suite for solo cello in C major," over an attractive scanned image. A recording of the piece plays until the user clicks on the image to go to the main menu.



Click anywhere to begin

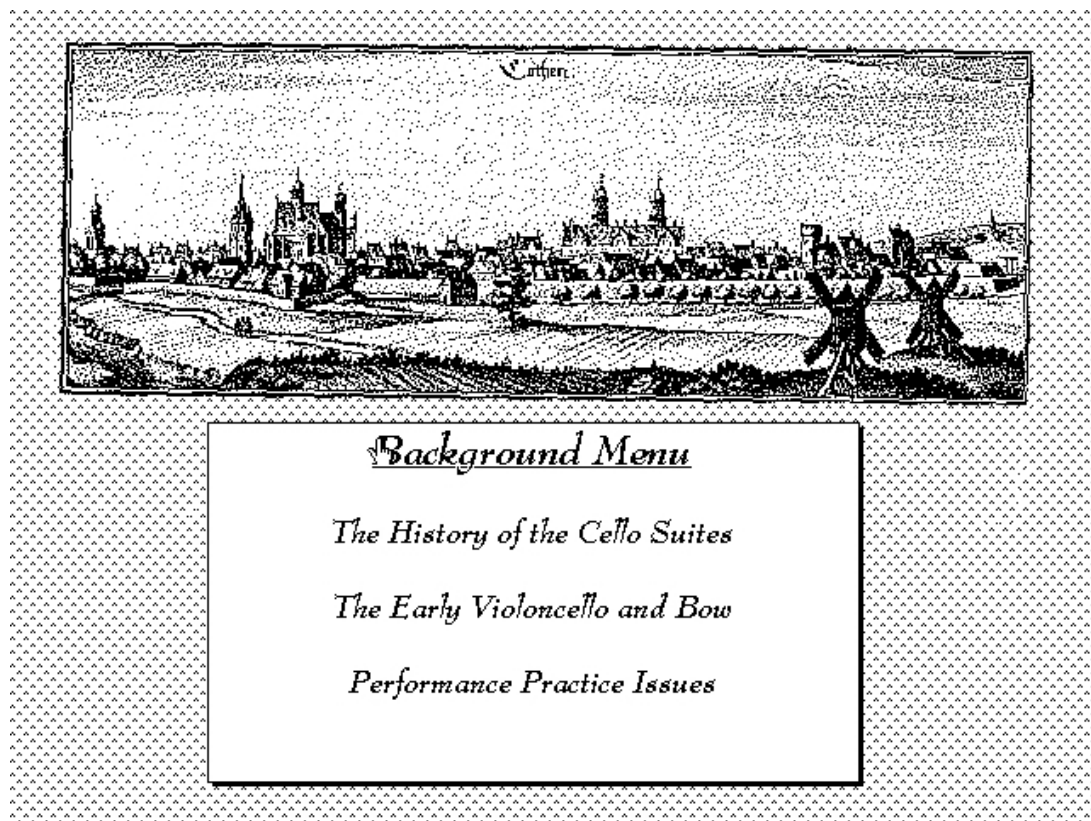
b. Main Menu

The main menu features sections on Background, Bowing Solutions, Dance Steps, View the Score, Editions and Performers, and Quiz.



c. “Background” Submenu

This menu features The History of the Cello Suites, The Early Violoncello and Bow, and Performance Practice Issues —under a scanned image of Coethen.




d. “Editions and Performers”

A short description of various historical and modern editions of the Cello Suites and information about the three performers featured in the stack—Casals, Starker, and Bylsma.

Editions & Performers

Editions:
Introduction
It is a pity that Bach's manuscript of the cello suites has been lost. Fortunately, there are four other 18th-century manuscripts available for our viewing.
Anna Magdalena
This is thought of as being the most authentic and most interesting of the manuscripts. It was done by Bach's second wife, probably between 1727 and 1731.
Kellner
This manuscript was possibly done around 1726, making it the oldest, by one of Bach's students. The manuscript is now held in the library at the University of North Carolina.
Westphal
This manuscript was done in the 2nd half of the 18th-century and is also held in the library at UNC.
Eppstein / Barenreiter
This edition is from the most recent Neue Bach Ausgabe and was published in 1988. It attempts to combine bowing strategies from all three of the above manuscripts.



e. Mock Quiz

A “place holder,” the quiz has a picture of Bach, and one question: Who is this man?

Quiz

Who is this man?

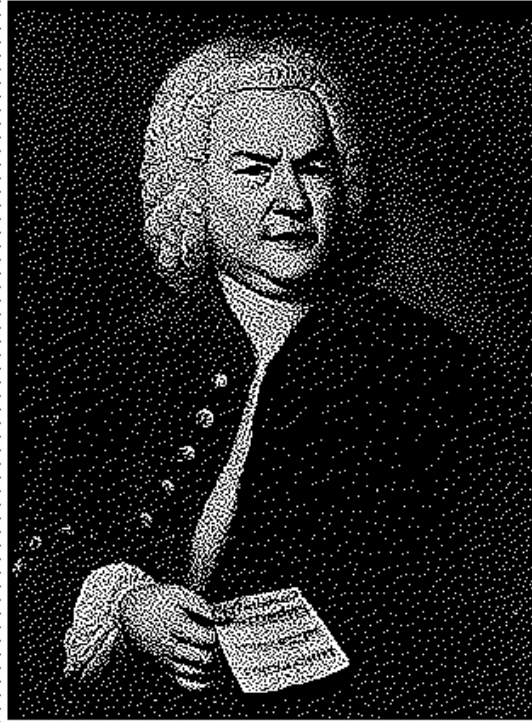









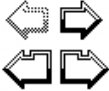
Figure 26. Fowler's "Bowing Solutions"

The movies play the recordings of by three performers that the user would hear by selecting the performer with a check-box and then clicking on Play This Measure. In the stack all music excerpts are implemented by accessing a custom-made CD through Voyager external commands.

Allemande - m. 1

<p>Anna Magdalena</p> 	<p>Eppstein / Barenreiter</p> 
<p>Kellner</p> 	<p>Starker / Peer International</p> 
<p>Westphal</p> 	<p>Gaillard / Schirmer</p> 





Measure #

Play This Measure

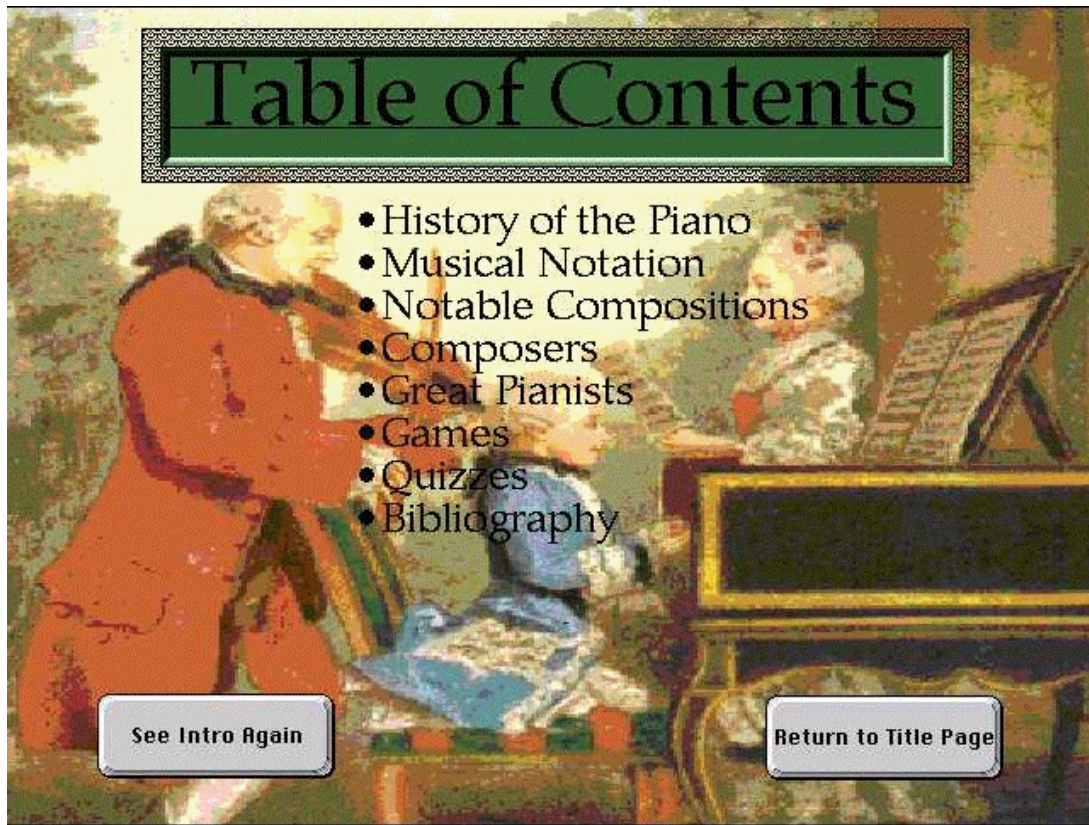
☐ Pablo
 ☐ Janos
 ☒ Anner

Bowing Solutions

Figure 27. Hynes's Stack

a. Main Menu

The main menu—History of the Piano, Musical Notation, Notable Compositions, Composers, Great Pianists, Games, Quizzes, and Bibliography—is set over a scanned painting of young Mozart at the keyboard.



b. History Submenu

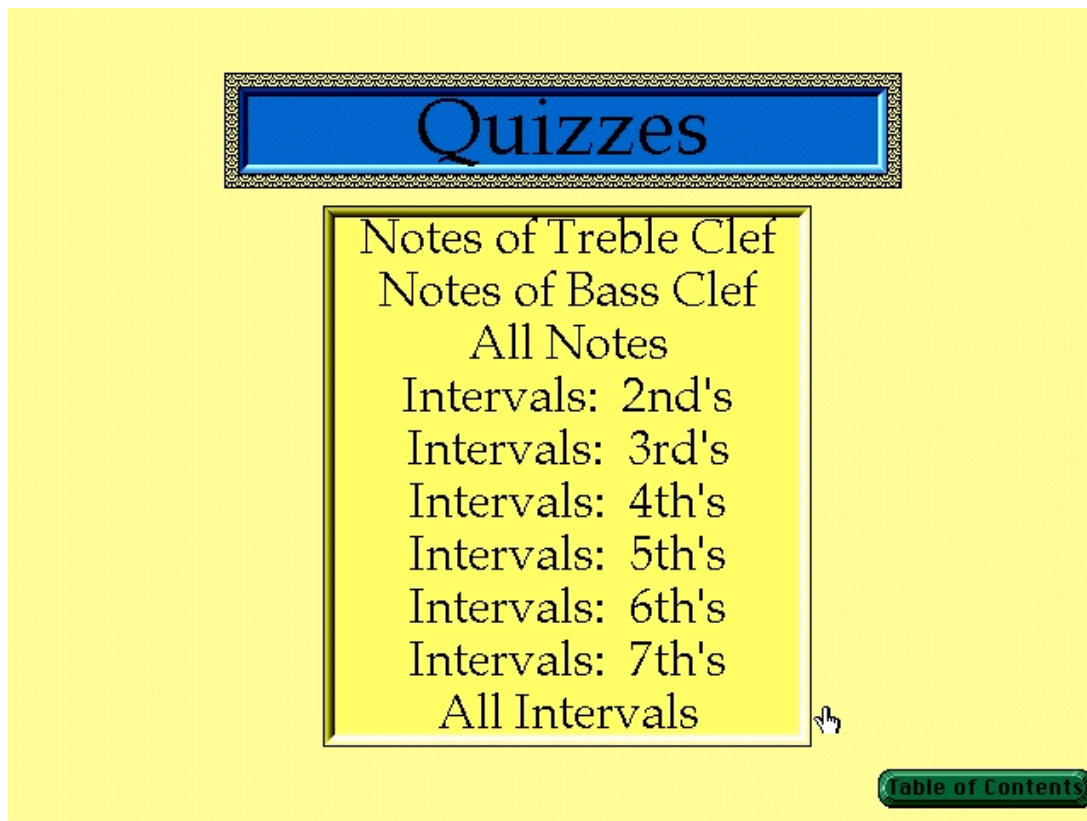
History of the Piano submenu includes the following selections: In the Beginning ..., Germany, Austria, England, United States, and Steinway & Sons. Each menu is set over a beautiful painting in full color.



c. Subsection Format



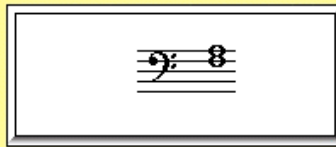
d. Quiz Submenu



e. Interval Drill

Notated intervals appear in a field on the top of the card. The student identifies the interval by clicking on buttons. Only the buttons representing intervals currently being tested are enabled. The music notation for the various intervals is stored on other cards, and copied to the quiz form by using HyperCard's Paint Tools under script control. Questions are presented in random order, and the student is given feedback.

Click button that identifies this interval.



Quit

- Diminished 2nd
- Minor 2nd
- Major 2nd
- Augmented 2nd
- Diminished 3rd
- Minor 3rd
- Major 3rd
- Augmented 3rd

- Diminished 4th
- Perfect 4th
- Augmented 4th

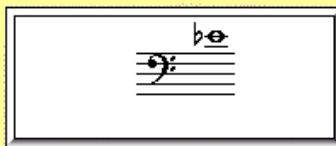
- Diminished 6th
- Minor 6th
- Major 6th
- Augmented 6th

- Diminished 5th
- Perfect 5th
- Augmented 5th

- Diminished 7th
- Minor 7th
- Major 7th
- Augmented 7th

Figure 28. Hynes's Music Reading Drill

Click on keyboard to identify this note.



Quit

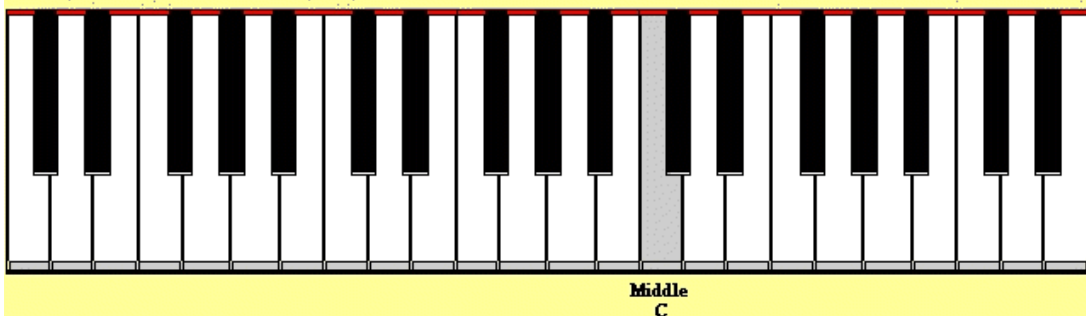
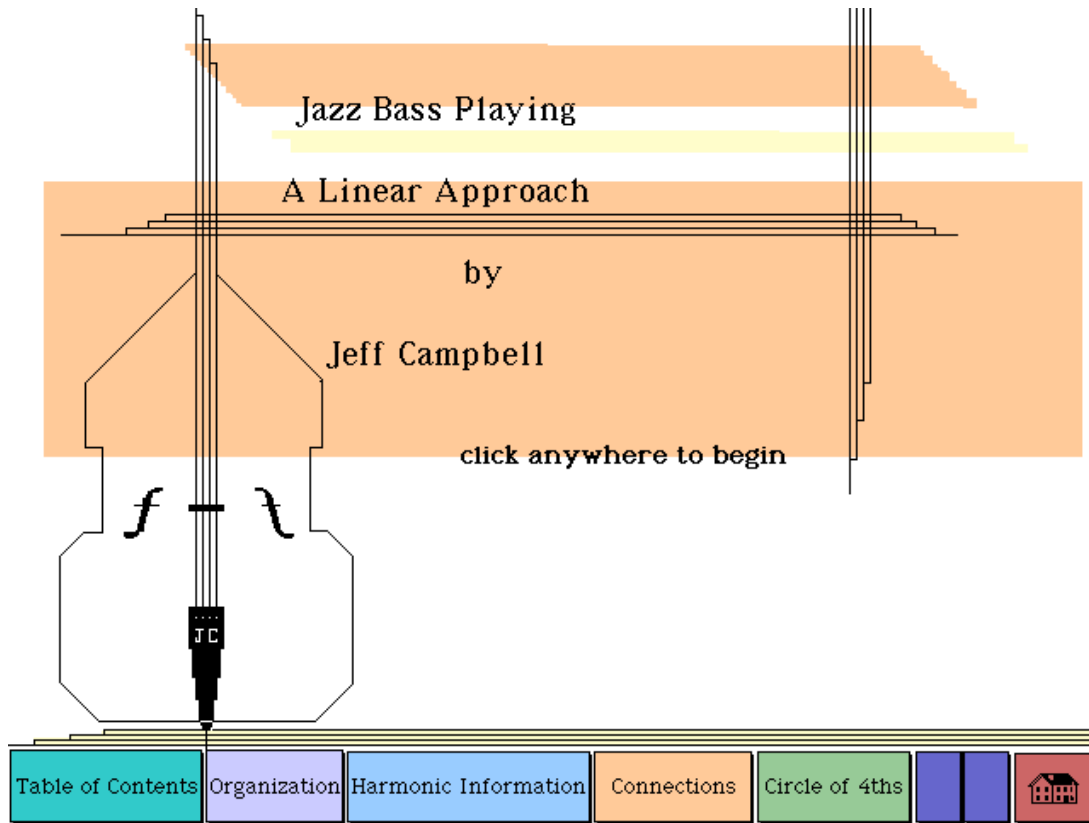


Table of Contents

Figure 31. Campbell's Stack

a. Title Page

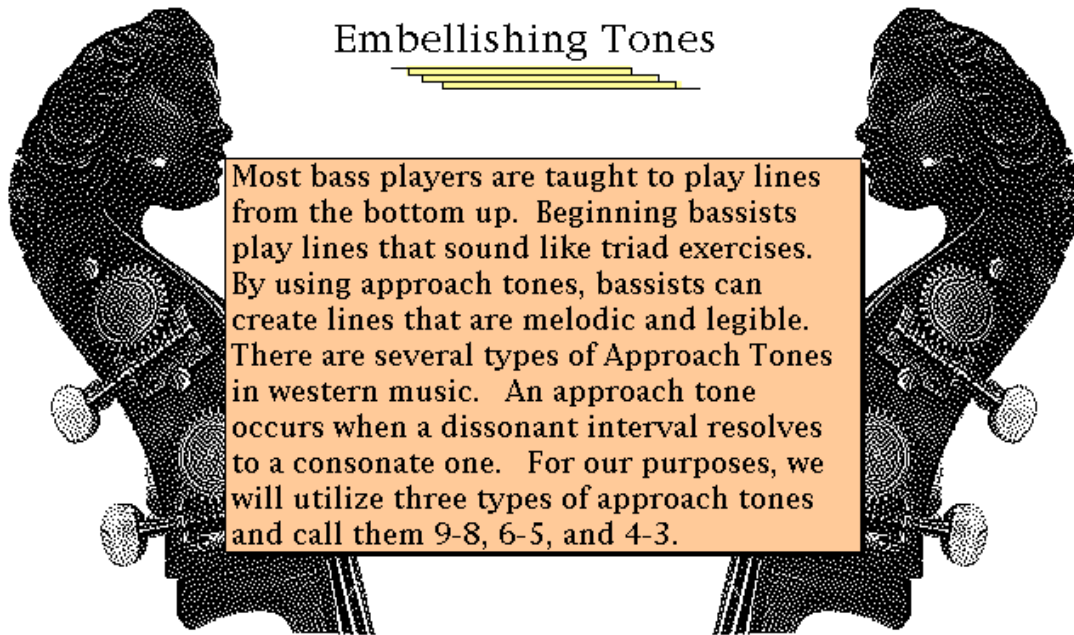
The title, “Jazz Bass Playing: A Linear Approach” is set over stylized line drawing of a bass. Buttons on the bottom of the card take the user to Table of Contents, Organization, Harmonic Information, Connections, and Circle of 4ths. The entire stack uses color and bass imagery effectively.



b. Embellishing Tones

A text field explains Campbell’s concept of “approach tones”— non-chord tones that resolve to consonant tones in 9-8, 6-5, and 4-3 linear patterns. The field is framed by a scanned picture of an ornate bass scroll, which is shown in symmetrical mirror image on either side of the field.

Embellishing Tones



c. Combining Lines

This frame shows how a three-voice rendition of a jazz chord progression can be implied in a single bass line. The progression is shown in open score with chord symbols. Below this is a bass line implying all three voices. The movie controller below the gif image plays a recording of Campbell performing the line on bass (from a CD he recorded to go with the stack). In the actual stack, the user would hear this by clicking on a Play button.

If the harmonic rhythm were doubled, it would give the player more time to combine all three lines into one line.

Cm7 F7 Bbmaj7 Ebmaj7 Am7b5 D7b9 Gm6/9

Cm7 F7 Bbmaj7 Ebmaj7 Am7b5 D7b9 Gm6/9

Play

Pause

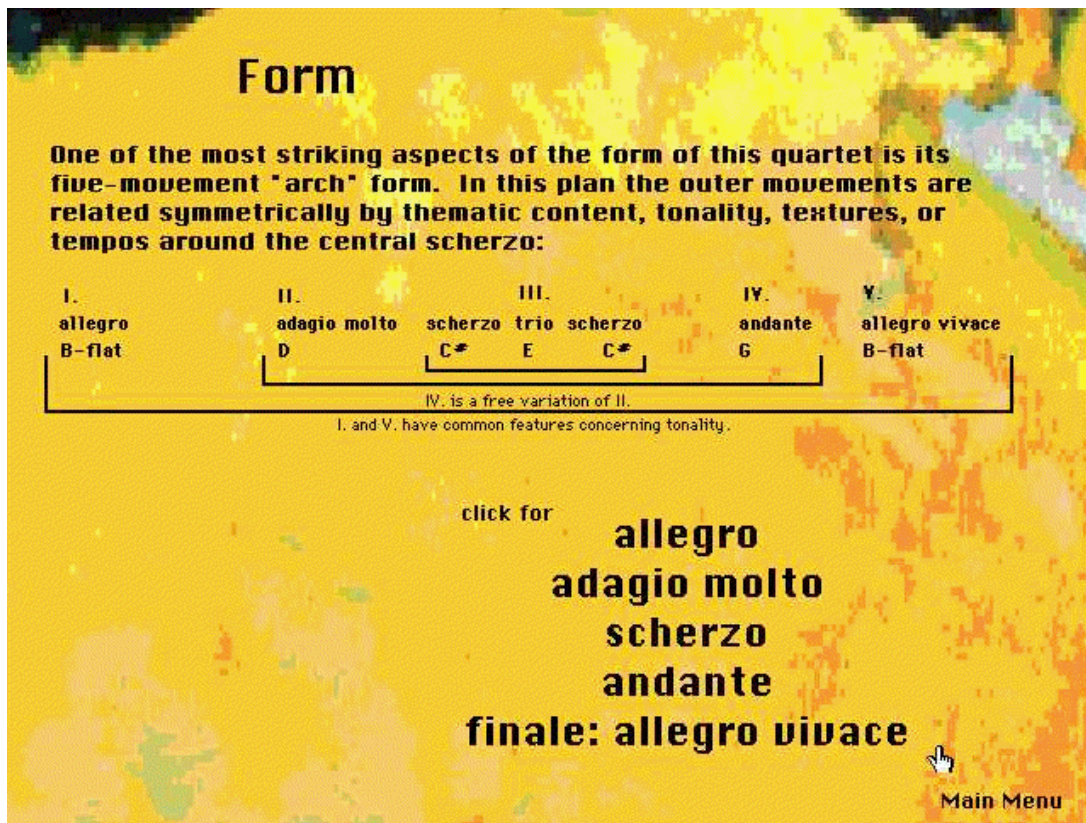
10/10

Table of Contents Organization Harmonic Information Connections Circle of 4ths

Figure 38. Leong's Stack

a. Form Menu

This card has a form diagram showing the arch-form in the quartet and a menu linked to each of the movements.



b. Interactive Form Chart

A form table is linked to the CD so that clicking on a line plays the appropriate section.

allegro

click to hear

section	bar	"tonality"
EXPOSITION		
	1 theme 1	Bb
	14 appendix to theme 1	Bb
	25 transition	C
	37 allusion to theme 1	
	44 theme 2	D
DEVELOPMENT	59	E
RECAPITULATION		
	133 theme 2, loosely inverted	F#
	147 transition, loosely inverted	Ab
	160 theme 1, loosely inverted	Bb
CODA	177	Bb

stop

Form Menu
Main Menu

c. Listening Menu

A menu linked to separate stacks for listening to each movement with running commentary.

Listening

click for

allegro
adagio molto
scherzo
andante
finale - allegro

Main Menu


d. Listening Section

The Listening section for the first movement. Described in the text above.

Exposition: theme 1

an attention-grabbing opening with instruments in unison texture on repeated B-flats (including one melodic major second between B-flat and C)

the motive is contracted to fewer... and fewer... repeated notes



section	tonality
EXPOSITION	
theme 1	Bb
appendix to theme 1	Bb
transition	C
allusion to theme 1	
theme 2	D
DEVELOPMENT	
E	
RECAPITULATION	
theme 2, loosely inverted	F#
transition, loosely inverted	Ab
theme 1, loosely inverted	Bb
CODA	Bb

[Play Through](#)

[Play This Card](#)

[Listening Menu](#)

[Main Menu](#)

1

