I, Dianne Michiko Sod, hereby submit this as part of the requirements for the degree of:

Master of Design

in:

the School of Design

It is entitled:

Design and Implementation of a Modular Preschool Multimedia Program

Approved by:

Karen Monzel, Chair
Dennis Puhalla
Janet Bohren
Marty Plumbo
DESIGN AND IMPLEMENTATION OF A
MODULAR PRESCHOOL MULTIMEDIA PROGRAM

A thesis submitted to the
Division of Research and Advanced Studies
of the University of Cincinnati
in partial fulfillment of the
requirements for the degree of
MASTER OF DESIGN
in the
School of Design
of the
College of Design, Architecture, Art, and Planning

2001

by
Dianne Michiko Sod
Ph.D. University of California, Los Angeles, 1993
A.B. Cornell University, 1987

Thesis Committee:
Associate Professor Karen Monzel, Chair
Professor Dennis Puhalla
Associate Professor Janet Bohren
Assistant Professor Marty Plumbo
ABSTRACT

This thesis details the design and implementation of a modular multimedia program, *ClickityKids for Preschool*. This educational program is designed to be integrated within the preschool classroom, supplementing and enhancing the existing curriculum while developing competency in five specific computer skills: gross mouse movement, directed mouse movement, mouse clicking, mouse pointing and clicking, and printing. In addition to its instructional benefits, *ClickityKids for Preschool* builds self-efficacy and positive attitudes for computer technology and fosters teamwork and cooperative play.

Based on a predetermined class schedule, the modular *ClickityKids for Preschool* program plays a different activity each week with minimal effort and virtually no supervision once it is launched. The program also provides great flexibility by accommodating multiple class schedules and allowing teachers to select an alternate activity if desired.

Each weekly activity, or module, is designed to be quickly mastered and completed, typically requiring less than 10 minutes to finish. Six representative modules are presented in this thesis: *Catching Seasonal Objects*, *Catching Shapes & Colors*, *Following Directions: Shapes & Colors*, *Connect-the-Dots: Letters*, *Alphabet Tennis*, and *Build-a-Picture: Halloween*.

Finally, classroom observations are described, difficulties encountered are reflected upon, and modifications and future directions are considered.
ACKNOWLEDGMENTS

I first want to thank my committee members: Karen Monzel, Dennis Puhalla, Janet Bohren, and Marty Plumbo. Their support and guidance began well before I started my thesis and has extended beyond the narrow scope of my design work. I feel fortunate to know them and honored that they are on my committee.

My graduate studies have been enriched and inspired by many DAAP faculty. Three instructors in particular deserve an added note of gratitude: Karen Cheng, Matthew Gaynor, and Carrie Nixon.

This thesis project would not have been possible without Kate Olsen, whose inspiration and grant proposal created the ClickityKids program. Kate's enthusiasm for preschool education is unmatched and I wish her luck in her multimedia endeavors.

Numerous friends and colleagues have been a positive influence during my graduate career. I especially want to thank: Chris Breyne, Chris Curran, Erin Lampe, April Mann, Jean Privett, and Tara Wyman.

Although scattered across the country, my family still manages to show me love and acceptance no matter what I'm doing. It goes without saying that I am grateful to be a part of this family!

Most of all, I want to thank my husband, Earl Sod. His encouragement, unwavering faith, infinite patience, and personal sacrifice allowed me to pursue and ultimately complete this design degree.
PREFACE

The work presented in this thesis describes software authoring using Macromedia® Director. References to programming and interface terms, as they relate to the Director authoring environment, are used throughout this document with the assumption that the reader is familiar with their meaning. Readers unacquainted with this application and its terminology should consult Appendix B: Glossary of Director and Programming Terms before continuing.
TABLE OF CONTENTS

List of Figures .............................................................................................................. iii
List of Tables ............................................................................................................... v
List of Flowcharts ...................................................................................................... vi

PART I: Background and Overview ........................................................................... 1
  1.01 Project Overview and Objectives ............................................................. 1
  1.02 Target Audience ..................................................................................... 3
  1.03 Target Environment ................................................................................ 5
  1.04 Additional Constraints ............................................................................. 6

PART II: Design and Development ............................................................................. 8
  2.01 General Considerations .......................................................................... 8
  2.02 Quick Starting Projector .......................................................................... 9
  2.03 Main Movie: Module Selection .............................................................. 10
  2.04 Main Movie: Navigation Bar .................................................................. 13
  2.05 Module Standardization ........................................................................ 18
  2.06 Module Printing .................................................................................... 19
  2.07 Module Teacher Information ................................................................. 21
  2.08 Module 1. Catching Seasonal Objects ................................................... 22
  2.09 Module 2. Catching Shapes & Colors .................................................... 25
  2.10 Module 3. Following Directions: Shapes & Colors .................................. 28
  2.11 Module 4. Connect-the-Dots: Letters ..................................................... 31
  2.12 Module 5. Alphabet Tennis ................................................................... 34
  2.13 Module 6. Build-a-Picture: Halloween.................................................... 35

PART III: Implementation and Conclusions ............................................................ 38
  3.01 Classroom Observations ....................................................................... 38
  3.02 Technical Difficulties Encountered ........................................................ 39
  3.03 Changes and Future Directions ............................................................. 40
  3.04 Summary and Conclusion ...................................................................... 43
# BIBLIOGRAPHY

Books ........................................................................................................... 44
Periodicals .................................................................................................... 45
On-Line Sources .......................................................................................... 46
Multimedia ................................................................................................... 47

# APPENDICES

Appendix A: CD-ROM User’s Guide ............................................................. 48
Appendix B: Glossary of Director and Programming Terms ..................... 49
Appendix C: Additional Required Xtras ....................................................... 51
Appendix D: Custom Lingo Overview .............................................................. 52

<table>
<thead>
<tr>
<th>Appendix D</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>D.01</td>
<td>Main Movie</td>
<td>52</td>
</tr>
<tr>
<td>D.02</td>
<td>Catching Seasonal Objects</td>
<td>54</td>
</tr>
<tr>
<td>D.03</td>
<td>Catching Shapes &amp; Colors</td>
<td>56</td>
</tr>
<tr>
<td>D.04</td>
<td>Following Directions: Shapes &amp; Colors</td>
<td>58</td>
</tr>
<tr>
<td>D.05</td>
<td>Firedog Maze</td>
<td>60</td>
</tr>
<tr>
<td>D.06</td>
<td>Connect-the-Dots Series</td>
<td>62</td>
</tr>
<tr>
<td>D.07</td>
<td>Alphabet Tennis</td>
<td>65</td>
</tr>
<tr>
<td>D.08</td>
<td>Build-a-Picture Series</td>
<td>67</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 1. A preschooler develops her computer skills while concentrating on completing the module. ............................................................3

Figure 2. The preschool classroom environment with the ClickityKids program in use. ...............................................................................6

Figure 3. Letterforms of the slightly modified Vogel Wide Bold font. .................................................................................................8

Figure 4. Screenshot of the splash screen from the projector. .........................10

Figure 5. Main movie start option icons........................................................10

Figure 6. Sample preference and reference list text file contents........................11

Figure 7. Screenshot of the Main movie entry screen........................................11

Figure 8. Screenshot of the Main movie schedule page. ...................................12

Figure 9. Screenshot of the Main movie reference list screen.............................12

Figure 10. Main movie framework and navigation bar. ......................................13

Figure 11. Main movie navigation icons ..........................................................13

Figure 12. Stop confirmation box .......................................................................14

Figure 13. Animation sequences for the confirmation box buttons.....................14

Figure 14. Custom color hand cursors used within the modules..........................18

Figure 15. Screenshot of the print dialog box floating over the grayed background. ..................................................................................20

Figure 16. Screenshot of a representative Teacher Information page ..................22

Figure 17. Screenshots of game play from the Catching Seasonal Objects module. .................................................................................23

Figure 18. Children characters used to introduce the four seasons in the Catching Seasonal Objects module. .................................................24

Figure 19. Print screen for the Catching Seasonal Objects module........................24
Figure 20. Screenshots of game play from the *Catching Shapes & Colors* module...................................................................................................26

Figure 21. Screenshots of instructional animations that precede game play for the *Catching Shapes & Colors* module. ..................................................26

Figure 22. Print screen for the *Catching Shapes & Color* module. ..........................................................27

Figure 23. Firedog illustrations used in the *Following Directions: Shapes & Colors* module. ..................28

Figure 24. Screenshot of game play from the *Following Directions: Shapes & Colors* module. .........................29

Figure 25. Vehicle illustrations used in the *Following Directions: Shapes & Colors* module. .........................29

Figure 26. Print screen for the *Following Directions: Shapes & Colors* module. ..........................................................30

Figure 27. Screenshot of game play from the *Firedog Maze* module. This activity formed the basis for the *Following Directions: Shapes & Colors* module. .............................................31

Figure 28. Progressive screenshots of game play from the *Connect-the-Dots: Letters* module. ..........................................................32

Figure 29. Print screen for the *Connect-the-Dots: Letters* module. ..........................................................33

Figure 30. Player choices for the *Alphabet Tennis* module. ..........................................................34

Figure 31. Screenshot of game play from the *Alphabet Tennis* module. ..........................................................34

Figure 32. Print screen for the *Alphabet Tennis* module. ..........................................................35

Figure 33. Screenshots of game play from the *Build-a-Picture: Halloween* module. ..........................................................36

Figure 34. Sample print screen for the *Build-a-Picture: Halloween* module..........................................................36

Figure 35. Cooperative play observed in the classroom. ..........................................................38
LIST OF TABLES

Table 1. Name, developer, and description of required xtras........................................51
Table 2. Global variable list for the Main movie.................................................................52
Table 3. Partial variable list for the Catching Seasonal Objects module.............................54
Table 4. Partial variable list for the Catching Shapes & Colors module..............................56
Table 5. Partial variable list for the Following Directions: Shapes & Colors module.............58
Table 6. Partial variable list for the Firedog Maze module..............................................60
Table 7. Partial variable list for the Connect-the-Dots module series...............................62
Table 8. Partial variable list for the Alphabet Tennis module.........................................65
Table 9. Partial variable list for the Build-a-Picture module series..................................67
LIST OF FLOWCHARTS

Flowchart 1. Key Lingo sequence for displaying the class names list and determine which module to automatically play in Main. ................................................................. 53

Flowchart 2. Key Lingo sequence for controlling motion and capture of objects in the Catching Seasonal Objects module................................................................. 55

Flowchart 3. Key Lingo sequence for controlling the release and capture of colored shapes in the Catching Shapes & Colors module................................................................. 57

Flowchart 4. Key Lingo sequence for randomizing the placement of the Flash vehicles and colored shapes in the Following Directions: Shapes & Colors module. ................................................................. 59

Flowchart 5. Key Lingo sequence for controlling the movement of the dog within the maze, the activation of vehicle animations, and stage panning in the Firedog Maze module................................................................. 61

Flowchart 6. Key Lingo sequence for advancing the progress of the puzzle in the Connect-the-Dots module series. ................................................................. 63

Flowchart 7. Key Lingo sequence for displaying a dynamic line in the Connect-the-dots module series. ................................................................. 64

Flowchart 8. Key Lingo sequence for controlling the moving letter that is hit by the player in the Alphabet Tennis module................................................................. 66

Flowchart 9. Key Lingo sequence for shuffling the z-order of the sticker pile when any sticker other than the top one is clicked in the Build-a-Picture module series................................................................. 68

Flowchart 10. Key Lingo sequence for controlling the top sticker when clicked on in the Build-a-Picture module series................................................................. 69
PART I: Background and Overview

1.01 Project Overview and Objectives

In recent years, there has been an explosion in multimedia programs targeting toddlers and preschool aged children. A study by the Software Publishers Association in 1996 found that “…62% of parents who have home computers and children under the age of six had purchased educational software for their youngsters. Those children owned six software titles, on average.”\(^1\) Despite the proven marketability and virtual glut of software aimed at this age group, a survey of current preschooler multimedia programs\(^2\) reveals that developers have fixated on the home user; seemingly oblivious to the needs of the classroom user.

Existing software formats, while appropriate for a home environment, preclude their effective use within the preschool classroom. The main problems with these programs include: the inability for the teacher to control which activity gets played without direct supervision and/or intervention, steep learning curves, inappropriately long activity time requirements, activities that do not facilitate taking turns, and inadequate teacher support for curriculum integration.

This thesis details the design and implementation of a modular multimedia program intended specifically for preschool classroom use. Conceived by Kate Olsen,\(^3\) the

---

\(^1\) Andrew Trotter, ”Software for Preschoolers Makes Market Inroads,” *Education Week*, v. 16 (Dec. 11 1996): 5.

\(^2\) Refer to the Multimedia section of the Bibliography for a list of CD-ROM titles evaluated.

\(^3\) Kate Olsen wrote the grant proposal and arranged for the *ClickityKids* program to be implemented at the preschool. After laying the foundation, the author was brought in to design, illustrate, and program the work presented in this thesis.
ClickityKids for Preschool project consists of over 35 weekly modular lessons that supplement and enhance the classroom curriculum while developing critical computer skills. This program features brief, easy to learn activities that require minimal adult supervision. With the ClickityKids program, teachers choose which module to run each week and construct a schedule for their class. In this way, the teacher tailors the computer activity to correspond with his/her classroom curriculum. The ClickityKids program also accommodates multiple class schedules, an important feature in an environment where different classes share the same computer. Finally, clearly defined educational objectives and curriculum integration suggestions are provided for each module. The main program structure and six representative modular lessons are presented in this document.

The ClickityKids for Preschool program builds proficiency in five specific computer skills: gross mouse movement, directed mouse movement, clicking, pointing and clicking, and printing. The keyboard is not utilized as an input device by the preschool user. While developing computer competency, the program’s activities are designed to supplement classroom learning, including: matching, sorting, rhyming, counting, alphabet sequencing, and the recognition of colors, shapes, and letter pairs. It is also expected that students’ motivation to learn, attitudes toward technology, and self-efficacy will improve through use of this program.

This project was done in collaboration with Kate Olsen, funded in part by a grant from the Josephine Schell Russell Charitable Trust, and used by preschoolers at a local community preschool. It should be noted, however, that the ClickityKids for Preschool project has been significantly redesigned for this thesis. Compared to the program that was implemented in the classroom, the version presented in this document
incorporates added functionality, a completely new module (*Following Directions: Shapes & Colors*), and numerous modifications, additions, and new illustrations. Most of these revisions were made as a direct result of feedback from both preschool teachers who had used the original version and committee members.

### 1.02 Target Audience

In defining the target audience for the *ClickityKids* program, two distinct groups need to be considered. The primary group targeted are the actual users of the program, preschool children. The second target audience is represented by the adults caregivers who must “buy into” the project for it to be successful; this group is comprised of both the teachers and the parents of the preschool users.

The users of the *ClickityKids* program are preschoolers, children typically between the ages of 3 – 5 years old. The physical, mental, and emotional needs of these impressionable users needs to be considered in all aspects of the design. Sensitivity to learning styles, gender stereotyping, and physical and mental disabilities is critical. 

For example, preschoolers have an almost insatiable need for stimulation and to be entertained. Whenever possible, colorful animations and a variety of sound effects are used to keep the

---


5 Approximately 5% of the preschool students who used the *ClickityKids* program have special needs including autism, Down Syndrome, and speech development.
preschoolers interested in the module activity. Because a preschooler’s attention spans is short, each activity is keep extremely brief, requiring only 4 – 6 minutes to complete. Children at this tender age are just beginning their literacy acquisition, so all instructions and buttons require auditory counterparts. Because preschooler eye-hand coordination is limited, active “hot spot” areas are kept generous in size and clicking and dragging the mouse is avoided.

In his book *What Kids Buy and Why: The Psychology of Marketing to Kids*, Daniel Acuff describes four critical product leverage areas for 3 – 7 year olds. These areas include concept, process, characters, and context. Acuff explains that the concepts of a product for this age group should not be overly abstract or require sophisticated reasoning. The process or technical aspects of the program should be kept as simple and straightforward as possible, with a relatively slow pacing. Children in this developmental period respond positively to age-appropriate characters, especially animals. Finally, these children relate best to a product when the context, its time period and socio-geographical location, are kept in the present with familiar locales. The guidelines presented by Acuff were, for the most part, followed by this program. The observation that the out-of-context space ship and dinosaur puzzles were the most popular in the *Connect-the-Dots* series, however, demonstrates that these are not hard and fast rules.

Designing for the preschool user alone, however, is not enough; teachers and parents need to be convinced of the project’s value in order for it to be effective. In addition to their decision making power, the attitude toward the program that these adults express

---

will be quickly perceived and internalized by children. Aside from delivering first-rate educational multimedia, the most effective method for gaining the support of teachers is to provide quality technical support and to be responsive to their suggestions. Supplying teachers with descriptions of each modular lesson and giving suggestions on how to integrate the software into their curriculum is another useful strategy. Finally, the incorporation of a computer printout in each module can be an important tool in winning adult support. The printouts serve a dual purpose in this capacity. First of all, the printouts extend the instruction and allow the activity to continue into other areas of the classroom, such as coloring, cutting, and promoting discussions. Secondly, these printouts play a valuable role in promoting the program to parents when brought home and shared by the child.

1.03 Target Environment

*ClickityKids* was designed to be integrated within the preschool classroom. The classrooms where the program was implemented serve as the prototypical model of the target environment.

The *ClickityKids* program was field tested in 7 classrooms of the community preschool. Running in both morning and afternoon classes, a total of 15 classes and approximately 225 students used these computer modules during the 2000/2001 academic year. Because multiple classes meet in the same room at different times, the *ClickityKids* program needed to accommodate multiple class schedules.

A single computer activity station was integrated within each classroom, surrounded by other play areas. Care needed to be taken, therefore, to use audio that could be understood in a busy environment without creating a distraction to others in the class.
Each center consisted of a Windows® based computer (500 MHz Compaq® Presario) and inkjet printer (Epson® Color Stylus 980). Because keyboard input was not required for any of the module activities, the keyboard was sometimes removed by the teachers to leave the mouse as the only input device.

Preschoolers were not required to use the computer in any of the classes were ClickityKids was made available. Instead, computer activity was offered during open playtime and free choice time, approximately 45 to 60 minutes out of the 2.5 hour class period.

1.04 Additional Constraints

Perhaps the greatest challenge of this project was its timeframe and organization. For reasons beyond anyone’s control, implementation of the ClickityKids program ran concurrent with its development. Modules were completed literally days in advance of their use within the classroom. Exacerbating the situation, the project utilized multiple developers, often working simultaneously on different modules. Standardization, clear documentation, simplification, and continuous communication became even more important considerations under these conditions.
With only 2 teachers to a class of approximately 15 students, the program needs to run continuously with little to no supervision by the teacher. Turn-taking would have to be defined by clear endings to the activities and audio prompts to give someone else a chance to play.

In order to accommodate the rapid turn-around expected in a classroom environment, it was stipulated that the modules would continuously loop, automatically returning to the start of the activity whenever the end of the module is reached.

To maximize convenience, it was insisted that the ClickityKids program feature an ability to automatically determine which module to launch based on the computer’s system date and the class schedule.

Finally, the inclusion of a “sleep” function was requested by the teachers. In order to minimize distractions, the program needs to be hidden from view during periods when computer activity is not allowed. Because a child may be assigned the responsibility of “putting the computer to sleep,” a simple method of temporarily blacking out the screen and halting the audio is required. Obviously, an equally simple method of “waking the computer” would also need to be implemented.
PART II: Design and Development

2.01 General Considerations

Although most commercial educational software titles still design for 640 x 480 resolution displays, it was decided that the ClickityKids for Preschool program would be based on 800 x 600 screen resolution with 16-bit color depth (thousands of colors). It is believed that a sufficient proportion of the computers in the target environment would be able to accommodate these higher requirements.

All text used in the module activities and Main movie button labels use a modified version of the font Vogel Wide Bold. This sans serif font was selected for its simple, rounded forms that resemble the Zaner-Bloser style letter shapes often taught to preschoolers as they learn to recognize and write the letters of the alphabet.

![Figure 3. Letterforms of the slightly modified Vogel Wide Bold font. This font was chosen for use within the module because of its similarity to the letterforms preschoolers learn. Modifications to the font are indicated in red.]

In particular, the clean uppercase ‘J’ and the simple lowercase ‘a’, ‘g’, and ‘t’ made this font particularly well suited for preschool use. Only the uppercase ‘I’ and lowercase ‘q’ required slight modification using Macromedia® Fontographer 4.1.5.

---

Young children seem to have a natural affinity for cartoons and cartoon characters. Because children are primarily visual in their orientation to stimuli, they relate well to the bright colors and bold graphics of cartoons. This attraction is evident in the growing success of network cartoon offerings and the popularity of the merchandise that they spawn. Building on the visual appeal of cartoons, a vector-based illustration style with bright colors and simple graphics was used throughout the program. Original illustrations were generated with Adobe® Illustrator 9.0 and animated using Macromedia® Flash 5.

Voiceover audio was recorded and edited using Macromedia® SoundEdit 16, version 2. Without access to child voice talent, the author recorded her own voice and shifted the pitch up 1.225 times to give it a higher, more child-like quality. The only exception to this audio style is found in the Following Directions: Shapes & Colors module, where an unaltered, adult male voice was used throughout.

The graphic and sound assets were brought together into a cohesive multimedia project using Macromedia® Director 8.0. Director was chosen for its ability to develop a cross-platform product and for its powerful programming features.

### 2.02 Quick Starting Projector

Following the guidelines detailed in a Macromedia TechNote, a quick starting projector was utilized to launch the Main movie. Both a Macintosh® and a Windows® version of this projector was created. In addition to loading quickly, the 450 x 280 pixel

---

9 My thanks to Earl Sod for this contribution.
projector also serves as a splash screen for the *ClickityKids* program, briefly displaying the title and copyright information at startup.

![ClickityKids Preschool](image)

**Figure 4.** Screenshot of the splash screen from the projector.

This screen is displayed for a few seconds as the main movie is loaded by the quick starting projector.

2.03 Main Movie: Module Selection

Launched from the quick starting projector, the 800 x 600 pixel *Main* movie opens with a screen that offers the user the option of auto-starting a module based on a class schedule, or selecting a title from the reference list of all module titles.

Presented in a text box, the list of available class schedules is dynamically generated from external preference files located in the same directory. Clicking on a class name initiates the auto-play function for that class.\(^{11}\) The program determines which module to play by comparing the class schedule contained in the preference file with the reference list file and the computer’s system date. Three possible auto-start errors are checked before attempting to open a module: (1) the system date is earlier than the start of the class schedule, (2) the system date is later than the end of a class schedule, and (3) there is either no module scheduled to play that week or the scheduled module is missing.

\(^{11}\) See Appendix D.01, for an overview of the custom Lingo used to program the auto-play feature of the *Main* movie.
Each class schedule is stored in a preference file using the naming convention “prefXX.txt,” where XX is a two digit identifier. The preference file contains the class name, start date, and list of weekly modules numbers to be played.

The reference list maintains the relationship between module numbers and title data.

If an auto-play error is detected, the user is shown an appropriate error message then taken to the Schedule page for that particular class. From this page, an alternate module can be chosen from a list of titles or a different class schedule can be selected.

If no error is detected, the scheduled module is launched and the teacher can turn the computer over to the students.

A class schedule is based on its starting date and referred to by a unique name or identifier. The Schedule page for a class displays this identifier and start date in two text fields. A text box containing the titles and weekly play dates is also shown on this page. The module titles are read in from the preference file for the class and the weekly dates are generated dynamically using the class start date. Modules listed in...
the schedule that are missing are automatically grayed out and cannot be clicked. Titles of available modules provide user feedback by changing color, switching the cursor from the default arrow to a pointing hand, and producing a brief sound when rolled over.

Figure 8. Screenshot of the Main movie schedule page.

The currently selected schedule is displayed along with class name (the schedule identifier) and start date. Unavailable or missing module titles in the list are grayed out and inactive.

The option to modify the schedule is shown but is not currently an active option of the project.

As an alternative to the auto-play function, the user has the option of selecting a module from the reference list of all titles. The reference list is stored in an external text file and provides the link between the file name and the full module title. Like the Schedule page, titles of modules that are missing appear grayed out and are inactive while titles of modules that are present produce rollover feedback and launch the module when clicked. The Reference page is distinguished by the yellow color of its text box.

Figure 9. Screenshot of the Main movie reference list screen.

All module titles are listed with unavailable or missing modules grayed out and inactive.

The option to modify the reference list is shown but is not currently an active option of the project.
2.04 Main Movie: Navigation Bar

When a module is played, it runs over the top of the *Main* movie framework as a movie in a window (MIAW). The *Main* movie frame is 10 pixels wide along the left, top and right sides, and 50 pixels wide at the bottom for the navigation bar. By placing the navigation elements at the lower end of the screen and keeping the active areas relatively compact, accidental pressing of these buttons is minimized.

![Figure 10. Main movie framework and navigation bar.](image)

The *Print* button slides into view only when a *Build-a-Picture* module is being run as these modules represent a special printing case. The yellow background of the *Print* button calls attention to it.

The buttons provide appropriate user feedback via highlighting, cursor change, and audio of the button name upon rollover. All navigation buttons feature a modest drop shadow to convey a sense of dimensionality and communicate their clickability.

The navigation buttons include *Stop*, *Sleep*, *Start*, *Print*, *Teacher* (information page), and *Schedule*. The *Stop*, *Sleep*, *Start*, and *Print* icons are designed to be simple and meaningful for preschoolers, using shapes and symbols readily recognized by this age group. Although the *Teacher* and *Schedule* icons represent more abstract concepts, these icons and the sections they navigate to are designed for adult use only.

![Figure 11. Main movie navigation icons.](image)

Top: *Stop*, *Sleep*, *Start*. Bottom: *Print*, *Teacher* (information page), *Schedule*.
Stop Button

The *Stop* function uses the red octagonal shape of a stop sign as its icon. Placed in the lower left corner, a stop sign in this location is a familiar carryover from the ubiquitous KidDesk® application. Clicking the *Stop* button initiates exiting the program.

When first implemented, clicking the *Stop* button resulted in immediately quitting the program. It soon became evident, however, that this arrangement suffered from accidental clicking during module play and premature exiting of the application.

To alleviate the disruptive effects of unintentionally exiting the program, a confirmation box was added. Successful activation of the *Stop* button causes a modal stop dialog box to appear above the grayed background image of the stage.

![Figure 12. Stop confirmation box.](image)

This modal dialog box appears once the *Stop* button has been clicked. Animated nodding and shaking heads as well as audio rollovers aid children in choosing the desired response.

Animated nodding/shaking heads and audio rollovers convey the meaning of the confirmation box’s button choices to preliterate preschool users.

![Figure 13. Animation sequences for the confirmation box buttons.](image)

These illustrations show the five images used to generate the animated nod and shake of the Yes and No buttons.

Although the stop dialog box appears to float above the module window like a MIAW, it is actually a different frame within the *Main* movie score. To accomplish this effect, a screen grab is first taken of the *Main* movie and any playing module window. This
image is then used in another frame of the *Main* movie as the background for the confirmation box graphic. Using this strategy minimizes the memory requirements of having multiple MIAWs open at once while providing the illusion of a modal, custom dialog box.

Even with the addition of a stop confirmation screen, the interruption of module play was deemed unnecessary and unacceptable as the *ClickityKids* program is intended to be started and stopped primarily by the teacher. As an initial solution, the *Stop* button was completely removed from the navigation bar during module activity runtime. However, with the *Stop* option inaccessible from the play screen, an overly circuitous route was required to exit the program; thereby reducing the functionality and usability of the navigation.

Balancing the need to minimize game play interruption and restrict access to the *Stop* function, a compromise solution was devised. When the *Stop* button is clicked, an audio and visual prompt to "click again" signals that a second mouse click is required. If the cursor moves off the *Stop* button before it is clicked a second time, the button is reset so that it must be clicked twice more to be activated. It is important to note that double clicking the mouse can be difficult for a preschooler and is not a required skill in any of the modules. The two click input required to activate the *Stop* button, however, is not a true double click as there can be an indefinite pause between the two mouse clicks.

**Sleep Button**

The *Sleep* button, represented by a yellow crescent moon icon, is used to send the computer into a simulated sleep mode. When pressed, the screen appears completely
black except for the small word “Sleeping…” in the center of the screen. The word slowly fades in and out of view and the dots of the ellipses subtly animate to signal that the computer is still operating. A mouse click or any key press immediately results in restoring the module and navigation bar.

Start Button
A green circle with an arrow pointing to the right was chosen as the icon for the Start function. This simple icon resembles a green traffic light, complementing the Stop button icon. Clicking the Start button at any time returns the module to the start of its timeline.

Print Button
The Print button with its unique yellow background stands out from the other buttons in the navigation bar. This button only becomes visible during game play of a module from the Build-a-Picture series. Details of the Print function are described in the Module Printing section of this thesis.

Teacher Button
The Teacher Information button (labeled simply “Teacher”) uses an abstract icon, the letter ‘i’ enclosed in a circle, to symbolize the concept of information. Although the preschooler cannot be expected to correctly interpret this button’s function, it is immaterial since they are not intended to have access to its content. The restricted access of this button’s function is visually implied by its location in the lower right corner of the screen, noticeably isolated from the other navigation elements.

Entry to the Teacher page is limited not because of the information it displays, but because the user gains the ability to switch modules from this page. The Teacher
button restricts access by requiring additional keyboard input after clicking. When clicked, an instructional tab that reads “now press ‘control’ + ‘T’” slides over the button’s active area. If the appropriate keyboard input is detected, the program proceeds to the Teacher Information page. If the cursor moves off the instructional tab without the keyboard input, the tab slides out of view and the Teacher function is silently deactivated. This keyboard entry system was chosen over a password, which was deemed too cumbersome for teachers to remember and input. The control key was selected as one of the required keystrokes because it is present on both Macintosh® and Windows® based keyboards and would be especially difficult for a preschooler to find since it is commonly labeled “ctrl.” For added security, a second keystroke was also required for activation. The letter “T” was chosen as the second keystroke because it is the first letter in the word “teacher” and should be relatively easy to remember.

The Teacher Information page is described in detail in the Module Teacher Information section of this thesis.

Schedule Button

Once successfully navigating to the Teacher Information page of a module, the navigation bar displays a Schedule button in place of the Teacher button. This button takes the user to the Schedule page for the currently active class, where the user can select a different module to play. Once again, this button’s abstracted icon is one that a preschool child would not be unable to comprehend. In this case, since access to the button is restricted, not only are preschoolers not intended to use this button, but they would never be exposed to the icon to begin with.
2.05 Module Standardization

Every *ClickityKids* module is named using the convention *moduleXX.dir*, where the *XX* represents a two digit identifying number. The numbers and titles of the modules are stored in a reference preference file.

The module plays within the framework of the *Main* movie as a movie in a window. The 780 x 540 pixel module MIAW is positioned 10 pixels down and to the right of the top left corner of the *Main* movie.\(^{12}\)

Standard markers included in every module’s timeline include: *start*, *print*, and *teacher*. These markers correspond to the *Main* movie navigation buttons and are required for the Lingo communication between the module and *Main* movies.

Assets of each module are organized and stored in one of four casts: *Lingo*, *Graphics*, *Text*, and *Sounds*. Appropriate cast placement of media is critical as numerous Lingo references specify both the cast member and cast names.

Custom color 32 x 32 pixel hand cursors are used for mouse activity within the modules. The hand cursors include an open hand to indicate that the mouse is over an active area of the module, a closed hand for “grabbing” action, a pointing hand to indicate the mouse is over a clickable area of the module, and a drawing hand to indicate that a dynamic line is being generated to the mouse position.

\(^{12}\) Relative positioning of the module to the main movie was accomplished using Lingo derived from: Allis. “My stage is smaller than my monitor. When I open a MIAW, and set it’s position to 0,0, it displays in the top left corner of the monitor, not of the stage. Why?” *Macromedia Director TechNotes* [database on-line] (December 1994) available from [http://www.macromedia.com/support/director/ts/documents/-fmkb0630.htm](http://www.macromedia.com/support/director/ts/documents/-fmkb0630.htm); Internet; Accessed 20 August 2000.
Whenever possible, recurring illustrated children “characters” are used throughout the modules to provide continuity and appeal to the preschooler's affinity for cartoon characters. These characters are always balanced in gender and racial diversity and include children with physical disabilities.

Every module begins with a brief instructional introduction for the activity at hand. The instructions are kept age appropriate by utilizing easy to understand language, reading any text aloud, and incorporating animations as much as possible. Instructions can always be skipped by pressing the space bar.

As described previously, game play should last approximately 4 – 6 minutes, clearly communicate an ending, and automatically return to the start of the module upon completion of the activity.

2.06 Module Printing

Printing is an included component in every module as an important tool in classroom integration, content reinforcement, and program promotion. Whether an achievement certificate or a supplemental activity, the child generates a tangible product from every completed module.

In addition to the print screen graphics, a descriptive sentence and copyright information is appended to the printout. To simplify tracking and editing, this sentence is stored in a field cast member named “printText” in the Text cast of each module and is read in to a global variable (gPrintText) at the start of the movie. Because it does not change, the copyright information is kept as a local variable within the Main movie.
In order to minimize accidental overuse of the printer, the *Print* function is made available only when appropriate. In the case of most modules, printing is only allowed at the end of the activity. Upon completion of the game, a printable screen related to the module is presented along with an audio description. A modal print dialog box, then appears above the grayed background and the child is asked whether or not they want to print the screen’s picture. This dialog box is similar to the previously described stop confirmation box.

**Figure 15.** Screenshot of the print dialog box floating over the grayed background.

The grayed background is a screen grab of the Main framework and module MIAW. Because it is only a graphic, none of the navigation bar buttons are functional and the dialog box appears modal.

If the child selects the “Yes” option of the print dialog box, they are instructed to pick up their picture from the printer and the buttons are replaced with the word “Printing.” The module print screen, text string, and copyright information are then formatted and sent to the printer using the Printomatic Lite xtra. The importance of removing the buttons prior to engaging the printer cannot be overstated; the overanxious preschooler will typically click the “Yes” button repeatedly if given the opportunity. Once the program has finished spooling information to the printer, the movie loops back to the start of the module.

If the child selects the “No” option of the print dialog box, the module is immediately returned to the start of its timeline without engaging the printer. No response after 90 seconds triggers a timeout that functions as if the user had clicked the “No” option.
An exception to the print sequence described above is made for the *Build-a-Picture* module series, where printing is made possible anytime during game play. It is only for these modules that the *Print* button slides into view on the *Main* movie navigation bar. Once this *Print* button is clicked, however, the child is shown the exact same print dialog box described previously. The open-ended nature of the *Build-a-Picture* modules also requires a different response when the “No” button of the print dialog box is clicked. Rather than sending the module to the start of its timeline, the activity is returned to the state it was left in when the *Print* button was clicked. To track whether the current module is a *Build-a-Picture* case, a global Boolean flag (gSpecialPrint) is initialized to FALSE in the *Main* movie and reset to TRUE for *Build-a-Picture* modules. By reinitializing this global variable to FALSE whenever the *Main* movie’s *Schedule* button is clicked, only the *Build-a-Picture* modules need to reference this Boolean flag.

Finally, while the module printouts should ideally be attractive and colorful, the layouts must be mindful of the ink coverage required per page. Because of the high turnover and usage rate that can be expected within a classroom setting, ink cartridge replacement expenses and long printing times must be balanced against the desire for large areas of color.

### 2.07 Module Teacher Information

A *Teacher Information* page is included at the end of every module. Intended to be viewed by adults only, this page consists mostly of text, including: a description of the module activity and its educational benefits, a calling out of the special programming features, curriculum integration suggestions, and a checklist of the computer skills being developed.
Initially designed as the module’s opening screen, this informational page was moved to the end of the module after receiving negative teacher feedback. Teachers reported that once it was read, it was a nuisance to click through this page in order to reach the module activity. Since the modules are designed to play for an entire week, there were more days where the placement of this page was causing aggravation than there were days that it was serving a useful purpose.

2.08 Module 1. Catching Seasonal Objects

The *Catching Seasonal Objects* module develops directed mouse movement skills while teaching counting and the sequencing of the seasons. In addition to the primary educational objectives, cycling through the seasons assists in developing temporal-sequential organization.

Module game play starts with a Fall scene where the child is instructed to catch 6 falling leaves by moving the cursor over the target objects. As each leaf is caught, its image appears in the child’s collection area, the number caught is counted, and the score is updated. Successful completion of the Fall catching task leads to similar scenarios where the child catches 8 snowflakes in Winter, 10 butterflies in Spring, and 12 frogs in Summer.
Each season features two different versions of the object to be caught. In the case of Fall and Spring, these differences are obvious. For the Winter and Summer seasons, the differences are more subtle; a variation in snowflake shape and frog color saturation. Although none of the objects are incorrect to catch, teachers can encourage advanced users to catch only one version of the moving objects or to catch the seasonal objects in a sequence which creates a pattern in their collection area.

*Figure 17.* Screenshots of game play from the *Catching Seasonal Objects* module.

Notice that this module is unique in utilizing photographic backgrounds. The photographs more accurately depict the seasons and separate themselves from the cartoon objects.

Boy and girl character animations introduce and conclude each season’s catching game. These children illustrations reinforce the seasonal theme with their appropriate attire and, while highly simplified, depict children with disabilities and varied ethnicity.
Having caught objects from all four seasons, the child has the option of printing a black and white image which summarizes the module’s activity. This printout can then be colored in and, if desired, cut out.

The structure of the Catching Seasonal Objects module gives it a high replay-quality. The starting positions, motion, and speed of each seasonal object is programmed to be randomized; no two games are ever completely identical. Furthermore, the ability to selectively catch one variant of the object over the other expands game play for the more experienced user.

13 See Appendix D.02 for an overview of the custom Lingo used to program the Catching Seasonal Objects module.
2.09 Module 2. Catching Shapes & Colors

The *Catching Shapes & Colors* module develops directed mouse movement skills while teaching counting, shapes, colors, and differentiation. In addition to the primary learning objectives, sorting the differences between shapes and colors promotes visual processing.

The module is comprised of four catching games of increasing difficulty. In each game, the child is faced with a screen of randomly falling shapes (triangles, squares, circles, and hearts) of various colors (yellow, blue, red, and purple). In the first two games of the module, the child is instructed to catch a particular shape. Although the child does not sort by color, as a clue, the correct shape is always the same color and the incorrect shapes are never this reserved color. In the next two games, the child is told to sort according to both shape and color attributes. In these later games, all shapes can be any color, adding to the difficulty level.

Whenever a shape is caught, the illustrated child holding the score card responds by either smiling, or making a funny face for incorrect ones. Correct catches also register by coloring in the shape in the user’s collection area, triggering counting audio, and increasing the catch total displayed on the score card.

Unlike the *Catching Seasonal Objects* module, the objects in *Shapes and Color Catching* only fall straight down from one of five set horizontal locations. The more predictable nature of the shapes’ descent was a necessary feature since the child must be able to navigate around incorrect shapes.

---

14 Green is not included as a color option because colorblind children would have difficulty distinguishing between red and green.
Figure 20. Screenshots of game play from the *Catching Shapes & Colors* module.

The screen on the left shows a game where the child sorts by shape only. The screen on the right requires that the child sort by shape and color for a catch to be registered. Notice that the expressions of the child characters responds to correct and incorrect catches.

In order to facilitate shape identification, the shapes do not rotate as they fall. Rotation would cause further complications as a square would momentarily become a diamond when rotated. Teacher feedback did result, however, in programming the triangle shape to randomly fall with the tip pointing either up or down.

Before each game begins, an introductory animation shows a child character pushing the target shape onto and off of the screen. The animation sequence includes a description of the shape and instructs the user how many shapes they are to catch in each game.

Figure 21. Screenshots of instructional animations that precede game play for the *Catching Shapes & Colors* module.
At the end of each game in the module, the animated child holding the score card cheers and then moves off the screen. If the module’s objectives have been met without any incorrect catches, a special end game animation is triggered and the child is told that they have earned a “gold star” for that game. At the end of the module’s last game, the child is shown a screen summarizing their shape catching and showing how many gold stars, if any, were earned. The child is then shown an achievement certificate reflecting their performance and given the option of printing a hardcopy of this screen. All certificates praise the child’s completion of the module and the number of perfect games is reflected in its text and graphical treatment.

Although the catching objective for the four games does not change each time the module is played, this module still represents an activity that may be replayed often. The color, shape type, and speed of descent for each shape is randomized; thereby keeping game play fresh.\(^{15}\) Additionally, by tracking the child’s performance and recognizing perfect completion of the games, the child is encouraged to repeat the module until it has been mastered.

\(^{15}\) See Appendix D.03, for an overview of the custom Lingo used to program the Catching Shapes & Colors module.
2.10 Module 3. Following Directions: Shapes & Colors

Although this is currently the only Following Directions module developed, it could easily be expanded beyond Shapes & Colors to modules with similar scenarios that teach concepts such as letter recognition or vocabulary. As such, this module is presented as one in a possible series of future Following Directions activities.

The Following Directions: Shapes & Colors module develops directed mouse movement skills and teaches children to listen to and follow directions while identifying shapes and colors. Additionally, the module’s objective, to help a lost animal, encourages children to be kind while the identification of the various vehicles (and their drivers) uncovered in the maze can be used as a catalyst for dress-up and role playing activities.

In the Following Directions: Shapes & Colors module, Sparky, a Dalmatian firedog, is lost and needs to find his fire truck. Game play begins with Sparky placed in a maze with six different colored shape destinations. The shapes include triangles, squares, rectangles, and circles that are colored either yellow, red, or blue; No two shapes are exactly the same. The child is instructed to visit a specific shape in order to uncover a clue to the fire truck’s location. Visiting an incorrect shape triggers audio feedback that the shape is incorrect and repeats the directions. When the child guides Sparky to the correct shape, a vehicle appears (a mail truck, school bus, police car, tow truck, or dump truck), animates, then directions to the next shape are issued. This process is repeated until
all shapes have been visited in the specified order. When the last shape is visited, Sparky is reunited with his fire truck and the activity is completed.

The child guides Sparky through the streets by placing the cursor over the dog and moving the mouse along the maze. Should the cursor stray off the path of the maze, Sparky stops and sits down. Sparky animates to give the appearance of walking and flips right or left to face the direction of movement whenever following the cursor.

Figure 24. Screenshot of game play from the Following Directions: Shapes & Colors module.

The firedog follows the cursor without requiring the child to click and drag the mouse. The firedog follows the cursor as long as the cursor is over both the firedog and the maze path.

Figure 25. Vehicle illustrations used in the Following Directions: Shapes & Colors module.

Vehicles represented twice give an indication of the animation that occurs when triggered by the firedog. Sound effects and a voiceover commentary accompany each animation.
Whenever a vehicle animates, the execution of other Lingo commands is temporarily suspended. As a result, neither the hand cursor nor Sparky will follow the mouse movement while a vehicle animation is active. This pause in Lingo control can result in an unexpected and sudden change in location of the hand cursor at the end of the animation if the child has been moving the mouse in the interim. In order to minimize confusing the novice user, the BuddyAPI xtra is used to restore the cursor position over the firedog at the end of each vehicle animation.

Although the maze itself does not change, game play is kept varied by randomizing which shapes are shown in the maze and the order that the shapes are supposed to be visited. As a result of this randomization, the activity cannot be completed by memorizing a set path; the child is forced to listen to and follow the verbal directions.

Once the fire truck has been found, the child is offered a chance to print and solve a traditional paper-based maze. This simple puzzle features a similar scenario where the child must reunite Sparky and his fire truck.

![Figure 26. Print screen for the Following Directions: Shapes & Colors module.](image)

This maze, while featuring lines to define walls rather than pathways, shares a similar objective with the module activity.

This Following Directions module began as a module titled Firedog Maze. Originally, the maze was much more complex and extended beyond the screen’s viewable area,

---

16 See Appendix D.04 & D.05 for an overview of the custom Lingo used to program the Following Directions: Shapes & Colors module.
required panning to reveal more of itself in any given direction. Furthermore, the vehicles themselves were used to mark the destinations rather than colored shapes. In the *Firedog Maze* module, the objective was simply to find the fire truck; no other directions to visit the other vehicles were issued. In this version, randomized vehicle placement and multiple mazes were implemented to ensure game play variety.

![Figure 27. Screenshot of game play from the *Firedog Maze* module. This activity formed the basis for the *Following Directions: Shapes & Colors* module. Requiring panning to uncover hidden areas of the maze proved to be too challenging for the preschool user.](image)

The *Firedog Maze* module was overhauled not only because it lacked substantial instructional merit, but it was also determined that it suffered from usability issues. The combination of disorientation from panning and frustration from dead ends in the complex maze proved to be more than the average preschoolers could handle. The revised *Following Directions: Shapes & Colors* module address both of these problems by simplifying and shrinking the size of the maze to completely fit on one screen. Additionally, the educational value of the module has been significantly enhanced by requiring the child to follow directions and recognize shapes and colors.

### 2.11 Module 4. Connect-the-Dots: Letters

The *Connect-the-Dots* module series develops directed mouse clicking skills while teaching recognition, pronunciation, and sequencing of either letters or numbers. Both a counting and an alphabet version of the *Connect-the-Dot* activity was completed; the letter version of the module is presented here.
Figure 28. Progressive screenshots of game play from the Connect-the-Dots: Letters module.
Each row illustrates the development of a puzzle from unconnected dots to completed picture.
The finished puzzles becomes colored, then animate across the screen.

The objective of this module is to complete a “drawing” by connecting 26 dots labeled
“A” to “Z.” Starting with dot “A,” the child points and clicks the dot to start a line. The
child then clicks the next alphabetically ordered dot to anchor the line between the two
dots. Clicking on an incorrect dot at any time results in a prompt to “try again” and
highlights the correct dot as a hint. When the last line connection has been made, the
child is congratulated, the puzzle becomes colored, a brief animation plays, and the activity ends.

After completing the puzzle, the child is given the option to print a new connect-the-dots puzzle to complete on paper. This black and white puzzle can then be colored in by the child and, if desired, cut out.

![Figure 29. Print screen for the Connect-the-Dots: Letters module. After completing the puzzle, the child can color and cut out the dinosaur image.](image)

Although game play of this module consists of completing a single Connect-the-Dots puzzle, four different puzzles were created. The subjects for the puzzles were selected for their believed appeal to young children, with a balance of choices that might be favored by boys and others that may be favored by girls. Care was taken, however, not to convey gender stereotyping. In the end animation sequences, notice that a boy appears in the window of the space ship and a girl pops up as the engineer in the train. The puzzle that the child will see is randomly determined in order to add interest and an element of surprise.

---


18 See Appendix D.06 for an overview of the custom Lingo used to program the Connect-the-Dots module series.
2.12 Module 5. Alphabet Tennis

The *Alphabet Tennis* module develops directed mouse movement skills while teaching letter pair recognition, pronunciation, and alphabet sequencing.

The child begins *Alphabet Tennis* by selecting a tennis player character, either a girl or a boy. The tennis player is then placed on a court opposite a net filled with the uppercase letters of the alphabet. The child controls the tennis player’s vertical position by moving the mouse; no clicking or rolling over of the character is necessary. One by one, the uppercase letters fly across the court toward the player. When the player “hits” the letter, it is transformed to its lowercase equivalent and returned to the net, triggering the next letter to be released. The letters are released from the net in alphabetical order and a missed letter is continually replayed until it is hit. The game ends when the entire alphabet is converted to lowercase.

Completion of the Alphabet Tennis activity leads to a printable supplementary puzzle. Like the module itself, the printed activity’s primary educational objective is to increase recognition of upper and lowercase letter pairs. In the printed exercise, the child must draw a line connecting an uppercase letter in the left column with a lowercase letter in...
the right column. This objective is demonstrated by including the letter pair for “A” with a line pre-drawn between its letterforms.

![Figure 32. Print screen for the Alphabet Tennis module.](image)

The five letter pairs that the child matches are randomly generated, creating a different puzzle for every printout.

Both components of this module, computer play and the printed activity, share a randomizing element that varies every game experience. Although the letters are released from the net in alphabetical order, the slope and speed of every letter’s movement is randomly generated. The letters are released from the net in alphabetical order, the slope and speed of every letter’s movement is randomly generated.19 In the printed activity, the five letter pairs that must be matched and their relative locations are also randomized.

### 2.13 Module 6. Build-a-Picture: Halloween

The Build-a-Picture module series are digital variations of the classic Colorforms® game and develop directed mouse clicking skills and creative play. A total of 13 different Build-a-Picture modules were developed with themes that celebrate holidays (Halloween, Thanksgiving, Christmas, Chanukah, Valentine’s Day, St. Patrick’s Day, Easter, and Mother’s Day) and seasons (snowman, fall, winter, spring, and summer). The appropriate use of seasonal themes for the Build-a-Picture modules promotes recognition of the holidays and serves as a catalyst for classroom discussion. The Halloween module, featuring a pumpkin background, is shown in this thesis.

19 See Appendix D.07 for an overview of the custom Lingo used to program the Alphabet Tennis module.
For each module in this series, the child is presented with a relatively simple background and a selection of “stickers” to choose from. The child clicks on a sticker to pick it up then clicks on the background to place the sticker within the picture. There are no wrong choices and there are countless different pictures can be created. The stickers can be repositioned at any time, both in their location and in their stacking order within the picture, or returned back to the sticker pile.\textsuperscript{20} When satisfied with the scene they have created, the child can then choose to print a hard copy.

Unlike the other modules in the program, the \textit{Build-a-Picture} series represents a special printing situation which allows the child to decide when they are ready to print. Furthermore, if the child clicks the print button accidentally while playing this module,
clicking the “No” option in the print dialog box resumes play with the scene still in tact rather than resetting to the start of the module. This built-in forgiveness assures that children can easily return to their scene without having to reconstruct it. The scene is automatically reset after printing or by pressing the Start button.

With its seemingly unlimited possible combinations and colorful printouts, the Build-a-Picture modules were an instant favorite with the preschoolers. This module provides immediate visible results from creative manipulation of the stickers while removing the fear some children may have of criticism or artistic inability.21

In order to extend the variability of the constructed pictures, a second set of stickers was added with a layered tab interface. Because the novice computer user may not recognize the significance of the clickable tabs, care was taken to ensure that enough stickers are initially visible to complete a satisfactory picture. It is hoped, however, that the more experienced user (or even a child who accidentally clicks on the tab) will share their discovery with others in the class. Surely, a child who sees another student’s printout with stickers unavailable to them would inquire as to how it was done. Thus, the use of the tabbed stickers may provide an opportunity for both discovery and sharing in the classroom.

PART III: Implementation and Conclusions

3.01 Classroom Observations

At the time this document was written, the ClickityKids program had been used by a preschool for approximately 9 months. During this period, classroom observations were primarily made by Kate Olsen and relayed to the author. However, a few personal visits to the preschool and review of video shot by Olsen provided some first-hand opportunities to observe the program in use.

Of particular significance was the observation that children interact with each other while engaging in the program’s activities. One of the major concerns prior to implementing the ClickityKids program was whether learning on a computer would necessarily be a solitary process. Classroom observations of the program in use, however, quickly dispelled this concern. Although only a single child was in operation of the computer at any given time, several other students would typically be crowding nearby, equally engrossed in the action. These children, even though they were not in control of the mouse, were able to share encouragement, advice, and genuine excitement in the game play action. Allison Druin and her colleagues described a similar social phenomena from their research of children in multimedia environments:

Children naturally want to be with other children… no matter how much technology children are offered (for example one computer per person), they will consistently form groups around one piece of technology (a
computer, video game, etc). We saw technology as a bridge and a catalyst for children interacting with each other. If children are strangers to each other, technology is the icebreaker. If children already know each other, technology is the means to get to know one another better. Children generally do not create in isolation: they want to share, show, and use technologies with each other.22

In addition to its social benefits, this quality of the ClickityKids program promotes both enactive and vicarious learning while fostering teamwork and cooperation.

Finally, it should be noted that classroom observations revealed a high level of enthusiasm for the program and general acceptance and praise by the teachers. Perhaps the best indicator of the program’s success is its planned continued use beyond this first trial year.

### 3.02 Technical Difficulties Encountered

The greatest technical difficulty this project has faced has been finding a reliable printing solution. The printouts for the modules require the ability to format and print images and text on a single page. Currently, the only affordable software to control print formatting is the Printomatic Lite xtra. Unfortunately, this xtra lacks updated drivers to work for many common personal printers in a cross-platform environment; including many models of Lexmark® and Hewlett-Packard®. Although the preschoolers using this project were able to successfully print, a more universally stable solution needs to be found if this program is to find a larger market.

Perhaps the most frustrating problems encountered had little to do with the technology but were a direct result of the concurrent development cycle and multiple developer

---

communication. Because development was ongoing with implementation, changes in
the program structure were exceedingly difficult to coordinate. Often, this resulted in
requiring a more complicated programming fix to avoid having to revise modules
already loaded on the preschool computers. Additionally, version control proved to be
an ongoing headache.

3.03 Changes and Future Directions

Despite the undeniable success of the ClickityKids for Preschool program,
modifications in structure and additional features would improve its usability, flexibility,
and maintainability.

The first suggested change in structure would be to prevent the module from
automatically restarting whenever it reaches the end of the activity. With the current
continuously looping system, there is no assurance that a child will see the
introduction and instructions given at the start of the module. Although the Start button
allows navigation back to the start of the timeline, it is doubtful that preschool users
will realize that they have missed important information. A more satisfactory scenario
would have each module paused at a start-up screen and require user input to initiate
the actual start of the activity. To lure children to the computer, an attractor animation
could demonstrate the module’s game play after a specified timeout period. Any
mouse movement or keyboard input would return the program to the start-up screen
and prompt the child to start the module.

Because the developmental stages of preschool aged children can vary greatly,
designing a single module to accommodate all preschoolers is a difficult task. An
obvious solution is to offer varying levels of difficulty for each activity. A paused start-
up screen, as described previously, would allow a natural stopping point for the child to select a level of difficulty for the activity. For example, the *Catching Shapes & Colors* module could be made easier by slowing the movement of the shapes and eliminating the number of shapes that fall at any given time; a more difficult version could include additional shape possibilities such as rectangles and ovals. The benefits of implementing this multi-level system are two-fold. First, the program will be a more pleasant and effective learning experience if beginning users are less frustrated and advanced users are more challenged. Second, by offering a choice, the program empowers the child. Making choices not only gives preschoolers a sense of power over the circumstances and events in their lives, it also builds self-reliance.23

The ability to maintain the schedules and reference list while within the program is a valuable feature that needs to be implemented. Non-operational, place-holder buttons for these functions are currently located in the *Main* program screens. Ideally, this feature would allow the user to add, modify, and delete information stored in the schedule and reference files. At present, a text editor such as SimpleText or NotePad is required to maintain these external files.

Another future change to the program would move some or all of the *Teacher Information* page content to a higher level within the information hierarchy. With the current structure, this information is only available from within the currently playing module. When no module is playing, selecting an activity from the *Schedule* or *Reference List* pages must be based on the module title alone. Titles are not descriptive enough to allow a teacher to make informed selections. While a printed

---

Part III: Implementation and Conclusions

A teachers’ guide to the modules would begin to address this problem, there is no reason that this information could not be made more accessible from within the program. Brief descriptions that appear whenever a module title is rolled over is one possible solution. Further functionality, such as the ability to sort the modules by educational categories or necessary computer skills, could also address this issue.

The *ClickityKids* program needs a unified, cohesive visual style and consistent user interface. While the modules presented within this document are self-consistent, the modules developed by other parties were not. In addition to diluting the program’s identity, variations in visual style can present a distraction and cause confusion. Even more serious, user feedback events, such as rollover and cursor changes, were not standardized. At the very least, a detailed style-guide needs to be developed and strictly adhered to for all modules.

The most immediate need of this program is to undergo formal formative evaluation. Present design direction has been primarily based on field testing the product in the preschool classroom; relying on informal classroom observation and teacher feedback. Future evaluation through user interviews, rigorous observations, and outcomes testing would provide valuable insight and revision direction.

The format of the *ClickityKids* program lends itself to be easily extended for use by other age groups. A future kindergarten and elementary school version could develop computer skill that target areas such as keyboarding, double clicking, click and drag, multiple item selections, and advanced interface (such as drop down boxes, menus, scrollbars, etc.) manipulation. As the target user’s age increases, more complex instructional content and longer activity times could be implemented.
3.04 Summary and Conclusion

From concept to classroom, this thesis had documented the design and implementation of a unique, modular multimedia program. Innovative features of ClickityKids for Preschool include the use of a schedule to automatically launch a selected module each week, multiple schedule management capabilities, and quick, easy to use modules that require little supervision and facilitate turn-taking.

The modules in the program provide age-appropriate instruction while building mouse manipulation computer skills. The activities are designed to be replayed often without becoming boring and each module provides an opportunity to produce a printout that complements the computer exercise.

The creation of an interactive computer program that fulfills the needs of preschool children, their teachers, and their parents has resulted in instructional benefits, social interaction, and successful classroom integration. This program serves as a valuable blueprint for future versions that may target other age groups and computer skills.
BIBLIOGRAPHY

Books


**Periodicals**


On-Line Sources

Allis. “My stage is smaller than my monitor. When I open a MIAW, and set its position to 0,0, it displays in the top left corner of the monitor, not of the stage. Why?” Macromedia Director TechNotes [database on-line] (December 1994) available from http://www.macromedia.com/support/director/ts/documents/fmkb0630.htm; Internet; Accessed 20 August 2000.


**Multimedia**


Appendices

APPENDICES

Appendix A: CD-ROM User’s Guide

Although the *ClickityKids for Preschool* program is designed to run from the computer’s hard drive, it will run off the CD-ROM for most computer systems. To play *ClickityKids for Preschool*, insert the CD-ROM disc into your CD-ROM drive, open the CD-ROM directory, and double-click either the icon labeled StartMac.exe (Macintosh users) or StartPC.exe (Windows users) to launch the application. If playback performance appears sluggish, quit the program, copy the complete contents of the CD-ROM directory to the computer’s hard drive, and restart *ClickityKids for Preschool* from the hard drive.

Macintosh® System Requirements:

- G3 processor or higher
- 32 MB RAM (64MB recommended)
- 800 x 600 Resolution monitor with 16-bit color
- CD-ROM drive
- System 8.1 or later

Windows® System Requirements:

- Pentium® 200 MHz processor or higher
- 32 MB RAM (64MB recommended)
- 800 x 600 Resolution monitor with 16-bit color
- CD-ROM drive
- Windows® 95 or 98
Appendix B: Glossary of Director and Programming Terms

This section contains definitions for selected Director and general programming terms that are used elsewhere in this thesis. Many of these definitions contain descriptions of the term as they specifically relate to the ClickityKids program.

Boolean: data type that evaluates to either TRUE (1) or FALSE (0).

Bounding box: property of sprites that have a visible component (bitmaps, Flash, shapes, text, fields), it is the smallest defining boundary that encloses the entire sprite's pixels.

Cast: the collection of cast members, or assets in a Director movie. The movies in this project all contain four internal casts: Lingo, Graphics, Text, and Sounds.

Cast member: the media or assets that make up a Director movie. Stored in the cast, cast members can be scripts, text, Flash swf movies, bitmaps, sounds, etc. The cast members of the project are organized according to asset type within one of four internal casts for each movie.

Global variable: a variable that can be accessed by any script in Director (once declared in a global command), including scripts in different movies.

Handler: a Lingo function or procedure. Handlers always begin with the line “on handlerName” and end with the line “end.”

Local variable: a variable that is only accessible to the handler in which it is declared.

Lingo: programming language of Director.

Marker: name assigned to a fixed location, a frame, in Director’s score. Markers are used for navigation. Also called a label. Modules in this project all contain a minimum of three markers: start, print, and teacher.

MIAW (Movie in a Window): a Director movie that played in a window other than the stage. A MIAW must be called by another Director movie. The modules in this project all play as MIAWs.

Modal: a window that requires user input before allowing access to other windows.

Projector: a standalone, or executable, application program created from a Director movie. A separate projector must be created for each platform. Projectors for Macintosh® and Windows® are included in the project CD-R.

Registration point: the referencing location within any cast member with a visible component. This location is expressed as an X,Y coordinate as is used to position the sprite on the stage.

Score: window timeline that, while in authoring mode, displays which cast members appear on the stage at various times. The score displays this timeline as a linear sequence of frames.

Script: a Lingo cast member.

Sprite: a cast member, once placed in the score, becomes a sprite; a collective description that includes the cast member, where it is on the stage, which frame it is in, and other properties.

Stage: the primary Director window where the action takes place. The stage is the visible translation of the score.

Variable: a storage location for data.

Xtra: a Director extension that enhances or adds functionality. Xtras are developed by both Macromedia® and third parties. A list of xtras required for this project is given in Appendix C.
Appendix C: Additional Required Xtras

The projector that launches this project has been compiled without any xtras in order to optimize its load time. The xtras, therefore, are located in a folder named `xtras` within the same directory as the projector itself. In addition to the default run-time xtras that are included with the application, the following xtras are required to run `ClickityKids for Preschool`.

Table 1. Name, developer, and description of required xtras.

<table>
<thead>
<tr>
<th>Xtra Name</th>
<th>Developer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BuddyAPI</td>
<td>Magic Modules</td>
<td>Provides functions for dealing with Windows API. Used to reposition the cursor at the end of vehicle animations in the <em>Following Directions: Shapes &amp; Colors</em> and <em>Firedog Maze</em> modules.</td>
</tr>
<tr>
<td>FileIO</td>
<td>Macromedia</td>
<td>Allows for the cross-platform reading and writing of text files. Used to read preferences for class name, start date, schedule, and reference list of modules from external text files.</td>
</tr>
<tr>
<td>Printomatic Lite</td>
<td>Electronic Ink</td>
<td>Provides basic cross-platform printing functionality for Director. Used to format and print screen graphics and text at the end of each module. Print out includes print screen graphic, descriptive text, and copyright information.</td>
</tr>
<tr>
<td>ScrnXtra</td>
<td>Kent Kersten</td>
<td>Allows screen captures to be specified and saved as graphic cast member. Used to capture a screen grab of the active module and surrounding main movie frame. This graphic was used as the background for the stop confirmation and print dialog boxes.</td>
</tr>
</tbody>
</table>
Appendix D: Custom Lingo Overview

This section is not intended as a substitute for the actual Lingo code or formal programming flowcharts. The information contained in this Appendix merely outlines the considerations of each programming challenge and the strategy employed.

D.01 Main Movie

The global variables initialized in the Main movie are primarily used to store various information about the currently active module. The custom Lingo for determining which module to auto-play is outlined in this section.

Table 2. Global variable list for the Main movie.

<table>
<thead>
<tr>
<th>Globals</th>
<th>Description</th>
</tr>
</thead>
</table>
gModuleName | Two character string for active module number. Modules numbered less than 10 are given a leading "0."
gModuleWin  | Module window reference.                                                     |
gStartDate  | Starting date for active schedule. This date is generated by reading in the first line of the schedule preference file, formatted as MM/DD/YYYY. The date object is formatted as date(Year, Month, Day) and is used to determine the play date for each module in the schedule. |
gPlayList   | List of module data read in from schedule preference file. Each module data consists of a 2 digit module number (#modNum) and the module title (#modTitle). |
gRefList    | List of all module data in numerically sequential order with no start date. |
gSpecialPrint | Boolean flag indicating that the active module is a special printing case. This situation occurs in the Build-a-Picture modules, where the child will return to the module in the state that it was left before clicking the print button. All other printing modules are reset as they return to the start of the timeline. This global variable is initialized to FALSE at the start of the program and every time the Schedule button is clicked. In this way, only the Build-a-Picture modules need to reset this variable to TRUE. |
gPrintText  | Descriptive string that is appended to the printout. This sentence always contains a blank line for the child's name to be added and is stored in a field cast member in the module's Text cast. |
beginSprite (Class name list sprite behavior)

Initialize properties.
Find all text files in the current directory that have a file name starting with "pref." Read in and store the first lines from these files as a string, separating each line with a return. Display this string in a text list box.

mouseWithin (Class name list sprite behavior)

Provide user feedback when rolling over a class name. Use the pointToLine function to determine rollover line color and cursor changes.

mouseUp (Class name list sprite behavior)

Read in the entire text file that corresponds to the class name clicked.
Read in second line of preference file, the start date, and compare with system date. Check that there are no out of range errors between the system date, the start date, and the schedule.

\[ \begin{align*}
\text{false} & \quad \text{Cannot initiate auto-play feature.} \\
& \quad \text{Go to appropriate error page.}
\end{align*} \]

\[ \begin{align*}
\text{true} & \quad \text{Determine which module to play based on system date and schedule.} \\
& \quad \text{Check that the module is present in the directory.}
\end{align*} \]

\[ \begin{align*}
\text{false} & \quad \text{Cannot initiate auto-play feature.} \\
& \quad \text{Go to appropriate error page.}
\end{align*} \]

\[ \begin{align*}
\text{true} & \quad \text{Initiate auto-play of the scheduled module.}
\end{align*} \]

Flowchart 1. Key Lingo sequence for displaying the class names list and determine which module to automatically play in Main.
D.02 Catching Seasonal Objects

This section describes programming the object sprites to move in a random fashion and detecting when they are caught by a catching sprite. Resembling a standard Falling Objects game,25 this module features complex motion, indicator sprites that

<table>
<thead>
<tr>
<th>Table 3. Partial variable list for the Catching Seasonal Objects module.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Globals</strong></td>
</tr>
<tr>
<td>gCaughtScore</td>
</tr>
<tr>
<td><strong>Seasonal Object Frame Script Behavior</strong></td>
</tr>
<tr>
<td>pMotionType</td>
</tr>
<tr>
<td>pCatchTotal</td>
</tr>
<tr>
<td>pStartSpriteNum, pEndSpriteNum</td>
</tr>
<tr>
<td><strong>Seasonal Object Sprite Behavior</strong></td>
</tr>
<tr>
<td>pMode</td>
</tr>
<tr>
<td>pSpeed</td>
</tr>
<tr>
<td>pThisMemName</td>
</tr>
<tr>
<td>pCatchSpriteNum</td>
</tr>
<tr>
<td>pActiveAreaSpriteNum</td>
</tr>
<tr>
<td>pIndicatorSpriteNum</td>
</tr>
<tr>
<td>pCatchSound</td>
</tr>
</tbody>
</table>

display the caught object, and a catch determination based on the actual pixels of the objects rather than a set distance between object and catching sprite’s locations.

**exitFrame (Frame script)**

- Make sure score does not yet equal the total number of objects to be caught.
- If score is not equal to total objects to be caught, remain in frame. For each object sprite, 5% chance of sending it a message to initiate falling or sideways motion from random location at random speed.
- If pMode is active, call custom motion handler. If pMode is inactive, respond to frame script motion initialization message.
- Game is over, go to next frame in module.

**exitFrame (Object sprite behavior)**

- If pMode is active, call custom motion handler.
- If pMode is inactive, respond to frame script motion initialization message.

**customMotionHandlers (mFall, mMoveRight, mMoveLeft)**

- Move sprite. Check if the catching sprite intersects the object's active area AND is visible on the stage.
- If the object is not caught yet, check if the object is no longer on the active area.
- Object has been missed. Move object sprite off-stage and deactivate.
D.03 Catching Shapes & Colors

Although the shapes only fall in a straight vertical path, the Lingo for controlling this

game is very similar to the code for the Catching Seasonal Objects module. The

Table 4. Partial variable list for the Catching Shapes & Colors module.

<table>
<thead>
<tr>
<th>Globals</th>
<th></th>
</tr>
</thead>
</table>
gCaughtScore |Current number of correct shapes caught.|
gPerfect |List of Boolean flags, one for each game, to indicate no incorrect catches were made. Initialized to TRUE and reset to FALSE if an incorrect catch is made.|

<table>
<thead>
<tr>
<th>Falling Shape Sprite Behavior</th>
<th></th>
</tr>
</thead>
</table>
pMode |The current status of the shape. Set to either #waiting or #falling.|
pSpeed |How fast the shape moves across the stage. Random value sent from the frame script to the shape sprite.|
pOrigLoc |Starting off-stage location of the shape sprite.|
pColorList |List of possible colors. Initialized with colors: #FFFF00, #3366CC, #9966CC, and #FF3333.|
pCatchSpriteNum |Catching sprite’s number.|
pActiveAreaSpriteNum |Number of the rectangular sprite defining the active area of the stage.|
pIndicatorSpriteNum |First indicator sprite’s number. Indicator sprite changes color when a correct shape is caught.|
pRightColorVal |Correct color (in hex) for a catch.|
pRightSound |Sound effect signaling a correct catch|
pGameNum |Game number in the sequence of games.|
pGameMode |Game catching requirements. Set to either “shape only” or “shape & color.”|
pRightShape |Correct shape for a catch. Set to either triangle, square, circle, or heart.|
pRightColor |Correct color for a catch. Set to either yellow, blue, purple, or red.|
pWrongShapeSound |Sound signaling an incorrect shape has been caught. Sound is voice saying “Not a "correct shape name."”|
pWrongColorSound |Sound signaling an incorrect color has been caught. Sound is voice saying “Not "correct color name.""|
flowchart shown for this module, therefore, focuses on the programming which determines which shape and color the falling object will take rather than the game itself. Unlike the previous catching module, the falling objects randomly switch cast members and colors and incorrect catches are possible.

Flowchart 3. Key Lingo sequence for controlling the release and capture of colored shapes in the Catching Shapes & Colors module.
D.04 Following Directions: Shapes & Colors

Controlling the firedog’s behavior within the maze uses Lingo developed for the previous version of this module, Firedog Maze. With the exception of panning the stage, the Lingo described in the Firedog Maze section of this Appendix should be referred to for an overview of this process. The programming notes for randomizing the location of the Flash vehicle and shape sprites are outlined in this section.

In this module, six Flash vehicle sprites and six shape sprites are placed in consecutive sets of sprite channels. The sequence of directions always starts with the first Flash vehicle and first shape, then progresses by increasing the channel numbers. Because the Flash vehicles cast members are not randomized, the order of cars that the firedog visits is always the same. Since the shape corresponding to each vehicle and the vehicle placement within the maze is randomized, the child cannot know where a particular vehicle is placed.

Table 5. Partial variable list for the Following Directions: Shapes & Colors module.

<table>
<thead>
<tr>
<th>Firedog Sprite Behavior</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pFirstFlashNum, pFirstShapeNum</td>
<td>First channel numbers in the range of sprites that makes up the Flash vehicles and shapes at each destination. “First” refers to the lowest channel number. The Flash vehicles are placed in consecutive channels, then after a break, the shape sprites are placed in consecutive channels.</td>
</tr>
<tr>
<td>pChannelDiff</td>
<td>Number of channels separating the first shape and first Flash sprites: pChannelDiff = pFirstShapeNum – pFirstFlashNum.</td>
</tr>
<tr>
<td>pNumOfTargets</td>
<td>Total number of destinations.</td>
</tr>
<tr>
<td>pRightLocNumber, pRightShape, pRightColor</td>
<td>Holds the value of the correct destination number and its corresponding colored shape. The shape cast members are each named using the convention “color shape.” The values for pRightShape and pRightColor are determined by reading in the words in the correct shape’s cast member name.</td>
</tr>
<tr>
<td>pWrongTimer</td>
<td>When an incorrect destination is visited, audio feedback is given. This timer prevents the audio cue from continually playing while the dog remains at this location.</td>
</tr>
</tbody>
</table>
The following custom handlers are called by the firedog sprite. The first two handlers, mRandomShapeMembers and mRandomFlashLoc, are called only once at the beginning of the sprite. The last handler, mGiveDirections, is run at the start of the game and every time a destination is visited.

**mRandomShapeMembers (Firedog sprite behavior)**

Build a list of all available shape cast members.
For each shape sprite, randomly select a shape from this list and set its cast member to this shape. Once selected, delete the shape from the list so it can only be used once.

**mRandomFlashLoc (Firedog sprite behavior)**

Build a list of location coordinates for the shapes and the Flash sprites within the maze. These coordinates are read in from field cast members and are ordered to correspond to the same destinations.
Randomly select a position within the lists (same value for both lists) and set the shape and Flash sprite location to the coordinates at this position within the lists.
Repeat positioning for every shape and Flash sprite.

**mGiveDirections (Firedog sprite behavior)**

Play audio directions based on the color and shape values read in from the currently active correct shape.

*Flowchart 4.* Key Lingo sequence for randomizing the placement of the Flash vehicles and colored shapes in the *Following Directions: Shapes & Colors* module.
D.05 Firedog Maze

Except for the stage panning effect, the Lingo outlined here also forms the basis for the maze behavior for the *Following Directions: Shapes & Colors* module. This maze activity features controlling the firedog’s movement in the maze whenever the cursor is... 

Table 6. Partial variable list for the *Firedog Maze* module.

<table>
<thead>
<tr>
<th>Maze Frame Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>pLastTargetNum Last vehicle target sprite’s number. This vehicle is the final destination in the maze, the fire truck.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Firedog Sprite Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>pTargetSpriteNums List of vehicle target sprite numbers.</td>
</tr>
<tr>
<td>pFirstSpriteRange, pLastSpriteRange First and last channel numbers in the range of sprites that makes up the maze. “First” refers to the lowest channel number.</td>
</tr>
<tr>
<td>pCursorSprite Sprite that acts as the custom cursor.</td>
</tr>
<tr>
<td>pArrowSpriteNum Arrow sprite’s number. This sprite appears to indicate the direction of motion whenever the maze scrolls.</td>
</tr>
<tr>
<td>pLeftBounds, pRightBounds, pTopBounds, pBottomBounds Holds the horizontal and vertical values that make up the “edges” of the active area of the stage. When the firedog reaches an edge, the stage sprites (all sprites within the range defined by pFirstSpriteRange to pLastSpriteRange) gradually slide over to bring more of the maze into view.</td>
</tr>
<tr>
<td>pOffStageLeft, pOffStageRight, pOffStageTop, pOffStageBottom Sprite numbers for rectangular sprites placed offstage along each edge of the stage. When a Flash animation is triggered, it is verified that it is completely onstage before playing. If the Flash sprite overlaps any of these offstage sprites, the stage is panned prior to animation.</td>
</tr>
<tr>
<td>pSitMem, pWalkMem, pStandMem Cast members of the various illustrations of the firedog. These cast members are used to animate the firedog.</td>
</tr>
<tr>
<td>pAnimTimer Timer used to animate the firedog whenever it is rolled over by the mouse. Sprite’s cast member is alternated between pStandMem and pWalkMem.</td>
</tr>
<tr>
<td>pRolloverSound Sound of a barking dog, signaling rollover by the mouse.</td>
</tr>
<tr>
<td>pPastLocH, pNewLocH Previous and current horizontal location values of the firedog. These values are continuously updated and compared to determine if the firedog should be flipped to face left or right.</td>
</tr>
</tbody>
</table>
over both the dog and the maze path. The Lingo that it uses is adapted from code that requires clicking and dragging the mouse.26

Flowchart 5. Key Lingo sequence for controlling the movement of the dog within the maze, the activation of vehicle animations, and stage panning in the Firedog Maze module.

D.06 Connect-the-Dots Series

The custom Lingo for the Connect-the-Dots module series, both Letters and Numbers versions, uses the same handlers to control the activity. The main considerations include rearranging the dots to correspond with the randomly selected puzzle, advancing the puzzle appropriately each time a dot is clicked, and drawing a dynamic line from the dot to the mouse location.

Table 7. Partial variable list for the Connect-the-Dots module series.

<table>
<thead>
<tr>
<th>Globals</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>gPuzzleList</td>
<td>Dynamically generated list of puzzle cast members. Cast members must be Flash assets placed in the Graphics cast and named with the prefix “puz” to be recognized as a valid puzzle.</td>
</tr>
<tr>
<td>gDotCounter</td>
<td>Current dot number counter. The next dot to click has a value of (gDotCounter + 1).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dot Sprite Behavior</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pNumberOfDots</td>
<td>Total number of dots in the puzzle.</td>
</tr>
<tr>
<td>pFirstDotNum</td>
<td>First dot sprite’s number. “First” refers to the dot in the lowest channel number.</td>
</tr>
<tr>
<td>pLineSpriteNum</td>
<td>Dynamic line sprite’s number.</td>
</tr>
<tr>
<td>pFlashSpriteNum</td>
<td>Flash puzzle sprite’s number.</td>
</tr>
<tr>
<td>pThisDotNum</td>
<td>This dot sprite’s number.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dynamic Line Sprite Behavior</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pCursorSpriteNum</td>
<td>Cursor sprite’s number.</td>
</tr>
<tr>
<td>pDotSpriteNum</td>
<td>Current dot sprite’s number.</td>
</tr>
<tr>
<td>pFirstDotNum</td>
<td>First dot sprite’s number. “First” refers to the dot in the lowest channel number.</td>
</tr>
</tbody>
</table>
Each Flash puzzle has the coordinates for each of its dots stored in a similarly named field cast member. The coordinates are read in at the start of the sprite and are used to position each of the dot buttons.

The Flash puzzle consists of a series of frames corresponding to each step in completing the dot-to-dot picture. Each of these stages are identified with numbered markers that are referenced through Lingo.

**Flowchart 6.** Key Lingo sequence for advancing the progress of the puzzle in the *Connect-the-Dots* module series.
Once a dot is clicked, a dynamically drawn line appears. One end anchored on the dot, the line follows the movement of the cursor until another dot is clicked and the line becomes anchored on the next dot.

A single shape cast member is used to generate the dynamic line. In order to properly display a connection between dot and cursor, which end of the line is positioned at the dot and the direction of the slope (controlled by the lineDirection property) are conditionally determined. Notice that when the line should be completely horizontal or vertical, it is reset to be slightly offset from straight. This offset is necessary to prevent the line from momentarily becoming invisible.

**Flowchart 7.** Key Lingo sequence for displaying a dynamic line in the Connect-the-dots module series.
D.07 Alphabet Tennis

This section outlines the Lingo used to control the release and ball-like motion of the letters from the net. The programming resembles code used in a Paddle Bricks arcade game.  

Table 8. Partial variable list for the Alphabet Tennis module.

<table>
<thead>
<tr>
<th>Variables</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Globals</strong></td>
<td></td>
</tr>
<tr>
<td>gGirlTennisPlayer</td>
<td>Boolean flag indicating the girl tennis player has been chosen. Initialized to FALSE.</td>
</tr>
<tr>
<td><strong>Bouncing Letter (“Tennis Ball””) Sprite Behavior</strong></td>
<td></td>
</tr>
<tr>
<td>pSpeed</td>
<td>How fast the letter moves across the stage. This value is multiplied by point(pMoveX, pMoveY) and is only set once.</td>
</tr>
<tr>
<td>pMoveX, pMoveY</td>
<td>Number of pixels the letter is to move horizontally and vertically. This value is randomly determined (between 6 to 10 pixels for X and between –5 to 5 pixels for Y) for each letter.</td>
</tr>
<tr>
<td>pPlayerSpriteNum</td>
<td>Tennis player sprite’s number.</td>
</tr>
<tr>
<td>pFirstLettSpriteNum, pLastLettSpriteNum</td>
<td>First and last letter sprite’s numbers. “First” refers to the sprite in the lowest channel number, “A.” “Last” refers to the sprite in the highest channel number, “Z.” Letters must be placed in consecutive channels.</td>
</tr>
<tr>
<td>pBounceSound</td>
<td>Sound effect when the letter bounces against the top or bottom edge of the stage.</td>
</tr>
<tr>
<td>pLetterSound</td>
<td>Sound of the letter name. This sound is played when the letter is converted from upper to lower case.</td>
</tr>
<tr>
<td>pHitSound</td>
<td>Sound effect of the letter hitting the tennis racquet.</td>
</tr>
<tr>
<td>pOrigColor</td>
<td>Original color of the unconverted, upper case letters.</td>
</tr>
<tr>
<td>pHiliteColor</td>
<td>Color of the converted, lower case letters.</td>
</tr>
<tr>
<td>pCurrentLoc</td>
<td>Starting location of the currently active reference letter on the net. This value is used to place the “ball” letter on the net whenever a new letter is about to be launched and is used to determine the return slope whenever a letter is hit by the player.</td>
</tr>
<tr>
<td>pCurrentLetterNum</td>
<td>Channel number of the currently active letter.</td>
</tr>
</tbody>
</table>

Flowchart 8. Key Lingo sequence for controlling the moving letter that is hit by the player in the Alphabet Tennis module.
D.08 Build-a-Picture Series

For all Build-a-Picture modules, the sticker pile is dynamically reordered whenever a sticker is clicked, bringing the active sticker to the top of the sprite stack.\textsuperscript{28} Thus, while

Table 9. Partial variable list for the Build-a-Picture module series.

<table>
<thead>
<tr>
<th>Globals</th>
</tr>
</thead>
</table>
|gStickerCount| Number of stickers added to the picture. There are a finite number of sprites, in this case 20, set aside to act as stickers.

<table>
<thead>
<tr>
<th>Sticker (Non-Top) Sprite Behavior</th>
</tr>
</thead>
</table>
|pTopStickerNum| Top sticker sprite’s number. “Top” refers to the sticker in the highest channel number.

<table>
<thead>
<tr>
<th>Top Sticker Sprite Behavior</th>
</tr>
</thead>
</table>
|pMode| The current status of the top sticker. Set to either #waitOffStage, #active, or #placed.
|pOrigMember| Top sticker sprite’s original cast member.
|pOrigLoc| Top sticker sprite’s original off-stage location.
|pCursorSprite| Sprite number of the open hand graphic that acts as the cursor.
|pPointCursorMem, pMoveCursorMem| Cast members for the graphics used for custom cursor changes. These graphics correspond to a pointing hand and a closed (grabbing) hand.
|pPrintAreaNum| Number of the sprite that defines the active area of the stage.
|pBottomStickerNum| Lowest sticker’s number. “Lowest” refers to the sprite in the lowest channel number.
|pOffsetH, pOffsetV| Horizontal and vertical offset values of the mouse click from the sticker sprite’s registration. These values are used to move the sticker with the cursor without it “jumping” to where the mouse is.
|pGrabSound| Sound indicating a sticker has been picked up.
|pPlaceSound| Sound indicating a sticker has been placed on the active area.
|pMisplacedSound| Sound indicating a sticker has been placed outside the active area and has been “returned” to the sticker pile.

it appears that every sticker can be moved by the user, only the top sticker ever follows the cursor.

**Flowchart 9.** Key Lingo sequence for shuffling the z-order of the sticker pile when any sticker other than the top one is clicked in the Build-a-Picture module series.
Flowchart 10. Key Lingo sequence for controlling the top sticker when clicked on in the Build-a-Picture module series.