Getting Interactive With Authorware 4:

Building Simulations and Games

Lloyd P. Rieber
The University of Georgia — Athens

Lloyd Rieber
604G Aderhold Hall
Department of Instructional Technology
The University of Georgia
Athens, GA 30602-7144
USA

Version: January 20, 1998

Contents

Chapter 6: How'd They Do That? 269

Preface

Frankly, I had always been wary of writing a book like this one. I had come close to doing it many times in the past, but writing any "how to" computer book is full of perils. For example, once begun I felt sure that the software would become obsolete just at the moment I finished. But several recent events finally convinced me to do it.

The first concerns Authorware itself. This authoring tool has both a respectable history and relatively safe future — a rare combination in the computer industry. It has been used by developers since 1987 and the recent upgrade to version 4.0 is good evidence of Authorware's stability. Also, in 1994 a "real" version of Authorware was released specifically for individual educators at a reasonable price, called either Authorware Academic or Authorware Star (not to be confused with an earlier Authorware Academic that disappointed and confused many of us in education — a story I won't bore you with here). This meant that ordinary classroom teachers might be able to get a copy or two of Authorware for their classrooms, not to program courseware *for* their students, but to give *to* their students to design and program their own projects. In other words, a version of Authorware capable of competing with other fine authoring tools already well-placed in the classroom, such as HyperCard, HyperStudio, and Digital Chisel. Getting powerful tools like these in the hands of students is an exciting concept that I want to support.

Unfortunately, Macromedia and Prentice-Hall have recently decided to stop publishing Authorware Academic.

The term "courseware" generally refers to any instructional computer software — software intending to teach somebody something.

Second, as a member of the faculty here in the Department of Instructional Technology at The University of Georgia I am continually looking for ways to improve my teaching. Authorware is and will continue to be one of the most important tools learned by students here at UGA. Our courses expect a great deal from students, many of whom are juggling full-time jobs and families. For example, in one class I teach students are expected to learn Authorware from scratch well enough to produce a stand-alone piece of courseware in the span of about 8 weeks (with two weeks for field testing and documentation preparation). Not surprisingly, the best that most students can do is a narrowly designed tutorial, heavy in content but shallow in meaning and experience for the person for whom the software is intended. This book is meant to help students learn about the concepts of interactive design from the very beginning and in a much more flexible way than is possible in traditional

course-based instructor-led approaches. Also, our department has reconceptualizing the entire curriculum leading to a master's degree in instructional technology as our university makes the transition from 10-week quarters to 15-week semesters. Our hope is to restructure the multimedia development courses in such a way as to get students learning about interactive design in more authentic ways. To pull this off requires more adequate and flexible learning resources. This book figures prominently in this effort.

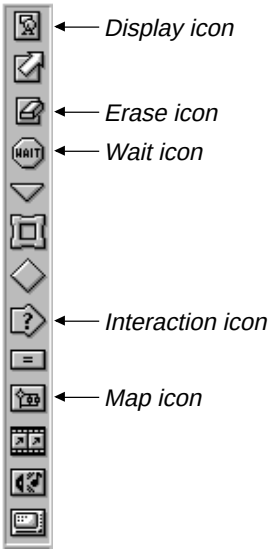
My final motivation for writing this book deals with my current interest in the concept of "learning by designing." I feel that the design process is a powerful mediator for learning. In a project called "KID DESIGNER," I have worked with elementary school children as they designed their own computer games about subjects they were learning at school. The games produced so far have been in science, language, mathematics, and mythology. These children did not learn Authorware. Instead, I worked for the children as their programmer. They designed the game goals and rules, created the graphics, wrote game questions and game directions, etc. and I put the pieces together using Authorware. (You can learn more about this project, and even download their games, by visiting the KID DESIGNER home page (the URL is "http://www.ncsu.edu/meridian/feat_1/kiddesigner.html"). In the next stages of the project, I hope to get high school students working together with the elementary school students. The idea is for the older students to handle the programming of the younger children's game ideas. For that reason, the projects in this book are also meant to be able to be completed by students as young as 15 or 16. The idea that Authorware could be used by such a wide range of ages for such different purposes speaks to the concept of "no floors, no ceilings" — good ideas for learning span many boundaries. Most anyone can begin using this tool and continue to use it in increasingly sophisticated ways.

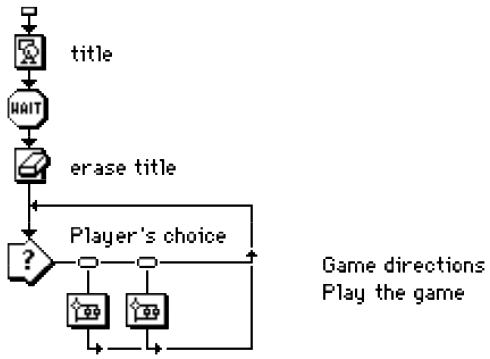
Prominent writers in this field include David Perkins, Seymour Papert, Idit Harel, and Mitch Resnick.

What is Authorware anyway?

The simplest answer is that it is a computer authoring tool designed for those who want to produce educational software. Despite the fact that the word "programming" has largely fallen out of favor, you essentially use Authorware to program computer software. Most writers today use the term "authoring" as a euphemism for programming. This is not without justification. In traditional programming languages, one has to learn commands (i.e. vocabulary) and the rules to use those commands (called syntax). Programming in Authorware is different. Instead of typing commands line after line, one creates a flow chart where each symbol performs a specific function. In Authorware, a flow chart is not just a tool for designing a program, it actually *is* the program! Consequently, more attention can be placed on the *design* of the program and less on the mechanics of programming.

For example, even if you have never used Authorware before, you can probably make some sense out of the following graphic:





You build flow charts like these simply by "dragging and dropping" icons from the palette on the left. For example, this little segment begins by displaying the game's title page. The program then waits until the player clicks the mouse button, after which the next icon erases the title screen. The next icon, an Interaction icon, allows the player to make a choice between reading the game directions and playing the game via two buttons that are automatically displayed on the screen when the program is run. The two Map icons are used to group or hold all of the necessary icons that make up the game directions and the game itself (including, probably, more Map icons). Notice how the flow chart clearly shows that the player will again be given these two choices after reading the game directions or completing the game (just follow the flow line as it exits each of the two Map icons).

Who should learn Authorware?

The "official" answer is that Authorware is intended for instructional designers and developers interested in creating computer courseware. My answer is that it is perfect for anyone wanting to take control of the computer to design their own creations. Yes, it is a great tool for teams of educators, subject matter experts, and programmers wanting to design instructional materials for students, but it's also a great tool for anyone who wants to enter the creative world of designing interactive computer software. It's also a great tool just to have fun with — it can be very satisfying to see one of your creations come alive on the screen. Of course, this takes time, patience, and perseverance. Some people insist that they cannot learn programming because they are not the "logical, analytic, left brain type." Actually, the "hard core, top-down, design everything first" way of programming is but one approach. The best designers I know are also highly creative and imaginative (and not all are adults — I have seen children as young as 10 use Authorware competently). In fact, I would argue that the so-called "right brain" attributes (like creativity) are far more important for anyone wishing to be a designer of educational software. The easy part is learning the "how to's" of programming, the hard

part is putting it to use. If you want to learn Authorware, are willing to invest some time and effort to do so, and won't bail out at the first sign of trouble, then I am confident that you *will* learn it. On the other hand, I'm equally convinced that you can talk yourself into believing that you can't. It's your choice.

The purpose of this book

This book is meant to help you to learn Authorware well in a minimum amount of time. Although one can learn to do some things very quickly with Authorware, it is still important to learn and master fundamental programming concepts, such as variables and functions, in order to go beyond creating "click and browse" courseware. Rather than saving these features for later, this book has you put them to use right away. Hopefully, you will begin to understand Authorware's rich collection of interactive features and capabilities from being immersed in their use at the very start.

This book uses what some refer to as the "project-based approach." You will learn how Authorware works by constructing a series of fully functioning projects. Each chapter presents one project designed to be completed in one or two sittings (1-2 hours). These projects are meant to be fun, intriguing, and, when finished, will act as working demonstrations of the most sophisticated levels of Authorware programming.

The project approach

The project approach simply means that one learns through building complete, stand-alone projects that are interesting and have clear goals and expectations. Good projects also have an easy to understand structure that makes sense at the start. Games and simulations make for good projects. One might say that this book was written much in the spirit of Norm Abrams on the American public television show "The New Yankee Workshop." Norm is a master carpenter who shows his TV audience how to build from scratch a piece of furniture or other woodworking project. Norm always starts by carefully choosing a project worthy of the effort, often visiting American colonial sites for inspiration. Once he finds a suitable subject, he then builds the piece twice — once for himself (plus to show the audience at the start what the finished product will look like) and again in front of the camera. The design that Norm follows usually includes many changes from the original, mostly to keep the construction within the reach of the amateur carpenter at home.

Likewise, all of the projects in this book were already existing products based on the dozens of games and simulations I have built for teaching, research, and just plain personal enjoyment over the last 16 years or so. The projects I chose for this book were among the most successful in helping others understand how to program with Authorware as well

as understand some fundamental concepts in designing simulations and games. It turns out they are also among the most fun to show and build. However, looking at some of the existing products made it clear that some compromises would have to be made. The main purpose of this book is, of course, to learn Authorware. Therefore, I needed to reconstruct the projects in a way that a novice could understand and build them quickly within one or two sessions. So, the chapters were written as I built the projects myself again from scratch, much like Norm in front of the camera. I was careful to record every keystroke and mouse click (with words and pictures) so that you could retrace my steps to build the same project yourself.

About the projects

Here are brief descriptions of the projects you will build (note that the simulation/gaming projects start with chapter 2):

Chapter 1: Slide Show. This chapter demonstrates how to use Authorware as a flexible and easy-to-use presentation tool; meant as a fast introduction to Authorware.

Chapter 2: Basketball Camp. A simple animated simulation of one of the fundamental rules of basketball defense: keep yourself between the ball and the basket. Good introduction to the use of data-driven animation.

Chapter 3: Mystery Number. A "guess the number" math game built with data-driven animation. (Remember playing the 'hot/cold' game as a kid?)

Chapter 4: Space Shuttle Commander. A simulation of Newton's laws of motion. Pretend you are piloting the space shuttle!

Chapter 5: Amazing Mazes. A template for games that use mazes; players have to answer questions as they proceed past numerous gates laid out on a maze background (programmed so that you can't cheat). Also includes directions on how to program a timer as a scorekeeping feature.

Chapter 6: How'd they do that? A collection of various interactive features you might like to have in your software. Examples:

- Allowing users to turn a program's sound on and off;
- Rotating screen objects by manipulating PICS movies;
- Building a "click and hold" interaction response type;
- Building "response palettes" that users can move freely around the screen;
- Creating "slide bars" that control data functions;
- How to save a program's data in a format that can be opened and analyzed by a spreadsheet.

Why the emphasis on games and simulations?

Starting with chapter 2, all of the projects you will build can be classified as either games or simulations (or both). There are two main reasons for the decision to focus on gaming and simulation. First, constructing a computer game or simulation requires mastery of many programming concepts. Despite the tendency to think of a game as "child's play," the programming of a game is among the most sophisticated of all computing projects. You'll be amazed at how much Authorware you will learn just by correctly programming the simplest of games. While games are demanding in terms of programming expertise, they are also at the same time fun to construct. You get instant feedback while you work. Once you know how a game is supposed to be played, you know within seconds if someone (or the computer) is "breaking the rules." Hence, you will know if your programming works just by playing the game (e.g. if a screen object goes through a wall instead of bouncing off of it as it should, you know where the programming "bug" is to be found).

But perhaps the most important reason for the emphasis on games and simulations is that I believe they remain among the most compelling learning environments yet devised. I think this is especially true for games. Through all my experiences as a teacher, researcher, and learner, I keep coming back to the creative use of gaming features as a potent learning strategy for all people. Well-designed games come as close as anything I have found to matching all of the complex requirements for a successful learning environment (including the cognitive, motivational, and social components). Arguments in favor of gaming can come from both the perspective of a teacher and student. The traditional strategy is to give students educational games as a learning approach. However, I find the idea of students constructing their own games to be far more compelling. Indeed, much of the current thinking in the field of instructional technology centers around "learning by building" or "learning by designing." Game design is perfectly suited to this point of view.

What this book won't teach you

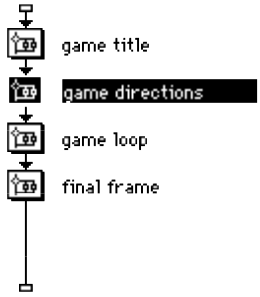
While no prior experience with Authorware is assumed, you are expected to already have much experience using a computer. This book will not teach you about general functions of the computer or other applications. In particular, you are expected to be completely comfortable with your computer's disk operating system (DOS), whether that be on the Macintosh (MacOS), or IBM-compatible (Windows). Although all necessary steps involved in manipulating files, icons, and graphics will be described thoroughly in this text, these explanations assume you already have experience with basic computing concepts (e.g. computer memory, file/disk management and security, etc.) and the graphical user in-

terface (GUI) techniques (e.g. “copying and pasting” within and between applications).

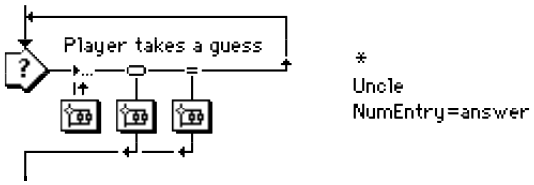
How to use this book

In each chapter, you will be asked to perform a series of steps to complete the project. Each step to be performed will be preceded by a check box, such as this:

☐ **Click once on the Map icon titled “game directions”:**



Follow the instructions carefully and check off the step when it is completed. This will help you from getting confused and disoriented. In the event that you make a mistake, these check boxes will help you to retrace your steps. Explanations and illustrations will accompany these steps. The illustrations represent what you should actually be seeing on your screen. You should strive to ensure that critical elements of illustrations match exactly what you see on your screen, such as the directions of the arrows showing the flow of a particular group of icons:



However, you will encounter lots of situations in which general features of illustrations will never exactly match what you see on the screen, such as the relative positions of windows. The graphical user interface (GUI) of Authorware results in all kinds of variations that have no bearing on the final result. Rest assured that the text will try to alert you to those times when absolute accuracy is required.

Of course, it is possible that you will follow and check off each and every step and still feel hopelessly lost at some point. When the inevitable happens, try not to panic. Instead, carefully retrace your steps starting at the last point where things seemed to be working. Try to approach these situations with the attitude of a creative problem-solver confronted with a challenge.

Look for some lighthearted commentary over here. I’ll try to give you some insights into my own ideas and experiences about design without boring you.

Finally, special instructions and reminders, plus informal commentary on design issues will be provided off to the side.

A word about platforms and versions

This book was written for Authorware version 4.0. If you are using an earlier version of Authorware (e.g. versions 2.0, 3.0 or 3.5), consult my web site for information about an edition of this book specifically written for those versions. Although the logic of the projects is essentially the same, there are significant differences in the interface design of version 4.0. It is also important to note that Authorware is available for both Macintosh and Windows. All of the illustrations in this book were produced using the Macintosh, however, they closely match what one would see on a Windows screen. Fortunately, differences between version 4.0 on the Macintosh and Windows are slight (the biggest difference being that Windows users need to press the "control" instead of "command" key).

Platform Alert!

Watch for this alert box for crucial differences between platforms.

Version 4 comes with significant improvements and enhancements. Examples include improved file compression (files are now typically 50-70% smaller than previous versions), improved cross-platform compatibility (a packaged file can be played on either platform), improved web authoring capability, improved support for image files (including GIF and JPEG files), enhanced ability to work with external content (such as graphics, sound, and movies), and compability with ActiveX controls, to name but a few.

An experiment in publishing

As any graduate student will tell you, books are overpriced. Of course, a publisher will quickly point out that books are expensive to produce (with paper the largest single cost). Authors will complain that their piddly royalties often don't justify the hard work. Surely, there must be a better way. I think there is and it's called the internet.

With this book, I am experimenting with the world wide web as a distribution outlet. More than likely, you yourself downloaded and printed this preface from my web site (http://itech1.coe.uga.edu/faculty/lprieber/authorwareinfo.html). If you were given a photocopy of the book to use (perhaps in a course you are taking), probably your instructor downloaded and printed it. (Indeed, the fact that you might be reading a photocopy of the book is no cause for concern — the quality is the same, if not higher, than in a "real" book.)

Transferring the costs and labor of printing to the end user allows me to offer this book at a low cost which is still fair to me as its writer. I also expect the internet to allow more access to the book by more people at the time they really need it (e.g. "Our development team just bought a copy of Authorware, now what?"). I also hope the electronic format of the book will allow me to continually update and improve the book. We'll see.

A little story (that you can safely skip)

When the Macintosh came on the scene in 1984, it quickly revolutionized many aspects of computing, most notably desktop publishing. Although it fast became the preferred platform for many people working exclusively with computer tools such as word processing and graphics, its use as a platform for computer-based learning environments was slow to catch on. At that time, the Apple II family of computers was the platform of choice for instructional software (drill and practice, tutorials, games, and simulations). The Apple II had been around for about 5 years and IBM had yet to make much of a dent in the educational market. The Apple II came with a built-in programming language — Applesoft BASIC. Granted, learning BASIC was no picnic, but it did allow a lot of people to become a special kind of "edutech" mutant — part educator and part computer hobbyist. It also allowed, for better or worse, the chance for elementary and high school students to get experience in computer programming (along with languages such as LOGO and PASCAL).

In the Macintosh world, in contrast, there just weren't any accessible means that an average Joe or Jane could do courseware development for the first few years of its existence. Ironically, the Mac's graphical user interface made programming unruly for everyone but the most dedicated and skilled programmers. Little was available for "just plain folks" like us. Colleges and universities that offered courses or curricula in educational computing tended towards labs of Apple IIe or IBM-PC computers for most course activities and clusters of Macintoshes for graphics, desktop publishing, and special demonstrations of emerging commercial courseware specially suited to its graphical interface. The advent of HyperCard (originally called WildCard) in 1986 was a significant milestone. With it, one could easily construct hypertext and hypermedia — textual and graphical documents in which users could explore in a nonlinear fashion. Most of the HyperCard software that still gets developed resembles easy-to-navigate databases of information in specific content areas. Of course, one can also develop educational games and simulations with HyperCard, but learning a significant amount of programming (called scripting) is necessary. When Authorware was first released around 1987, it represented something very new.

I first started using Authorware in 1988 when it was called *Course of Action*. It was the principal product of a new company (called Authorware, Inc.) headed by Michael Allen. Its creative use of a graphical programming environment based on an interactive flowchart was amazing to many of us. Although the term "multimedia" had yet to cause people's eyes to roll back into their heads, Authorware made the integration of text, graphics, animation, sound, and video virtually seamless. Most remarkable to me was the range and

depth of interactions that one could produce (with surprisingly little "programming" in the traditional sense). However, the first few versions of Authorware were slow and cantankerous. (The first version I used was very prone to crashing.) Implementing one's courseware was also awkward. You actually had to buy "student disks" from the company in order to distribute the software you created. Fortunately, the company soon shook off the quirkiness in both the product and their approach. Those who get into educational computing today and start with programming tools such as Authorware fail to appreciate its contrast with the programming tools available up to the time of its release.

Authorware and HyperCard were long viewed as tools in competition. This was never an accurate view. Both did different things well and other things poorly. If both were screwdrivers, one was a phillips and the other a flathead. HyperCard was designed for hypertext (with graphics) whereas Authorware was designed to build instructional courseware. Each could do the other, but only with a struggle. On pure economic grounds, there was no competition — HyperCard was literally given away by Apple with every Macintosh that was sold. Many people bought a Macintosh just to get HyperCard. It was truly a "Killer App"! Authorware, in contrast, was (and still is) extremely expensive, even for the academic community. Single copies of Authorware originally cost about $8000 for corporate customers. Higher education had it a little better —my university department shelled out over $2000 for our first copy back in 1988. Fortunately, prices have fallen substantially since that time.

Then a couple of things did and didn't happen in the early 1990's that gave Authorware a decided edge. First, Authorware was bought out by Macromedia. A lot of us had seen software tools come and go quickly on the Macintosh. Macromedia's purchase seemed to assure us that Authorware would be around for awhile. Plus, Authorware was now supported by a company with the resources and reputation for serious multimedia development (Director is one of their other well known authoring products). Next, despite persistent rumors to the contrary, a complete overhaul of HyperCard never happened. While different features were embedded in new releases, it essentially remained the same product as before. One of the most frustrating things about HyperCard is its handling of color — it continues to be a grayscale product with cumbersome color capabilities. In contrast, Authorware upgrades were fairly significant improvements (the latest upgrade to 4.0 is a good case in point). Another important difference was Authorware's commitment to cross-platform development and delivery. One can now develop courseware on either the Macintosh or Windows platform and convert to the other with little or no reprogramming (although serious cross-platform development still demands much planning). Authorware has kept up with

advances in the computing industry while retaining virtually the same interface that it had in 1987. Among the most notable and recent advances is its compatibility with Macromedia's ShockWave technology permitting courseware developed with Authorware to be distributed over the internet.

Authorware is not perfect, however. There are still lots of things that frustrate developers (besides its cost). One of the biggest pet peeves continues to be its slowness. Fortunately, 4.0 delivers much improved performance, especially on PowerPC computers. The wonderful interface of Authorware comes at the cost of consuming loads of the computer's processing capability leaving less available for the actual execution of the program. This can be a serious flaw for anyone interested in developing games and simulations for anything but the fastest Macs and PCs. There is also the misconception that one can pull Authorware out of the box and start developing sophisticated courseware immediately. I think this perception was actually promoted (inappropriately and perhaps unintentionally) by the company early on. Yes, in contrast to programming in BASIC, PASCAL, or C, the learning curve is slight. But believe me, there *is* a learning curve to Authorware. There are two consequences to this attitude. The first is just the general frustration of not being able to do the things promised right away. The second is the tendency to create courseware that merely scrapes the surface of what is possible with Authorware. With some devoted time and effort, one can create very imaginative and interactive courseware with Authorware. But there is no such thing as a free lunch here either — you get out of it what you put into it.

Starting with version 3.5, files produced with Authorware are able to run in "native mode" on PowerPC computers to take advantage of this computer's RISC chip, resulting in a significant improvement in speed.

Acknowledgments

Although the number of people to whom I need to express my thanks continues to grow, I'd like to thank several people here. First, I thank Simon Hooper, a friend and colleague at the University of Minnesota, for his helpful encouragement. Simon is also a creative Authorware user, teacher, and writer. I'd also like to thank someone I have never met — George Beekman. My first attempts at learning HyperCard were very successful mainly due to a book that George wrote called *HyperCard in a Hurry*. I found his approach to be very effective. The step-by-step project approach used here largely mimics his style.

I'd also like to thank the following individuals who patiently tried out early drafts of the chapters and provided me with valuable feedback: Brett Bixler, The Pennsylvania State University; John Braidwood, Huntly College, Hamilton New Zealand; Gail Fitzgerald, University of Missouri-Columbia; and Pam Miller, University of Pretoria, South Africa.

Finally, special thanks go to my daughter, Becky. In 1996 (when she was 17 years of age), she graciously served as the book's first beta tester. Her experiences and comments on early drafts of this book have been extremely helpful. She has also reinforced my idea that Authorware is a wonderfully creative tool for anyone to learn and should not be reserved only for professional designers and developers. Expert Authorware programmers don't have to have letters after their names.

LPR
January 20, 1998

Notes:

Chapter 1: Slide Show

Authorware as a Presentation Tool

This chapter introduces you to Authorware and shows how to create Authorware files from scratch. In this chapter you will build a slide presentation that is both "plastic" and "modular." You will easily be able to add, delete, modify, and rearrange slides in a matter of seconds.

This chapter is presented in 2 sessions. You will be saving the following files:

slide1.a4p
slide2.a4p

This chapter is meant to introduce you to the structure of Authorware files and will give you a simple example of most of the Authorware icons. This chapter is suitable as a thorough introduction to Authorware for a person who has never used it before.

I know, you want to get started right away building simulations and games. Why begin with such a mundane project as a slide show? Well, it pays to learn well some of the fundamental features of Authorware and also to master procedures that you want to begin taking for granted (such as constructing a simple display). Go through this chapter carefully and make sure you understand how each icon is used.

The Problem

You've been asked to give a presentation tomorrow morning on the role and benefits of simulations and games in education. The presentation will be in a large lecture hall and about 200 people are expected to attend. You will have access to a desktop computer connected to an LCD projector. While it might seem to make more sense to use a desktop presentation package such as Persuasion or PowerPoint, you know that you will want to build a presentation that will not only be useful tomorrow, but will also allow you to embed interactive examples you intend to create later. Maybe Authorware can help.

If you are in a hurry to start building simulations and games, at least go through this chapter up to the section "What's next?" on page 48.

Introduction

Before you begin using Authorware to build simulations and games, it makes sense to get acquainted with the basic features and structure of Authorware. This chapter assumes you have never used Authorware before. You will get introduced to most of Authorware's icons, however, don't expect every detail of every icon to be included. Experienced Authorware users can skip this chapter, though you might also want to complete it as a refresher of some basic concepts and procedures.

Although you might want to dive right into simulation and game design, this chapter will ease you gently into Authorware. It is important for your first attempts to be successful. A core principle of this book is that building useful projects is motivating and also makes it easier to understand. In that spirit, this chapter shows you how to use Authorware as a presentation tool. Even though this is a little like using a Mack truck to deliver a loaf of bread, you will walk away from the chapter having a useful tool for making and giving slide shows. One advantage of using Authorware in this way is that you can "copy and paste" the projects from the other chapters, when completed, back into this slide show as examples.

Frankly, if your goal was to use Authorware only as a presentation tool, you would be well advised to learn a desktop presentation tool, like Persuasion or PowerPoint, instead.

As described in the Problem section, the content of this slide show will help answer the question "Why Simulations and Games?" Perhaps you'll learn a few things about the benefits of games and simulations for education as a result.

Note: The significance of the "xtras" folder is explained in Appendix A.

Before you begin, be sure that Authorware has already been installed on your computer's hard drive along with its various resources. At the very least, check to be sure that the Authorware application, the appropriate runtime application (Macintosh: RunA4M.68k, RunA4M.ppc, or RunA4M.fat; Windows: RunA4W32.exe) and a special folder called "xtras" are all installed in the same folder. A reminder, too, that you are expected to be comfortable with the basic operating concepts and procedures of your computer, including either MacOS or Windows. For example, you should already be comfortable with the concept of "copying and pasting." If you find yourself asking either "Where is my file? I thought I saved it!" or "How did my file wind up there?" you probably need to spend some more time understanding the file structure of your operating system.

Macintosh Alert!

There are three different runtime applications for the Mac due to the inherent differences between PowerPC and nonPowerPC computers. Authorware 4.0 runs in "native mode" on the PowerPC, so use the ".ppc" version if you know you will only be using this system. The "fat" version can be used on both systems, though this runtime application is quite a bit larger than the others. (More about this in Appendix A.)

This chapter will use several multimedia resources, such as graphics, sound, and digital movies. It is suggested that you load the "Rieber Clip Media" folder onto your computer's hard drive before beginning this chapter so that you will have these resources handy when they are needed. This folder can be downloaded from the book's web site. Here's the URL:

http://itech1.coe.uga.edu/Faculty/lprieber/authorwareinfo.html

Don't worry, it's free.

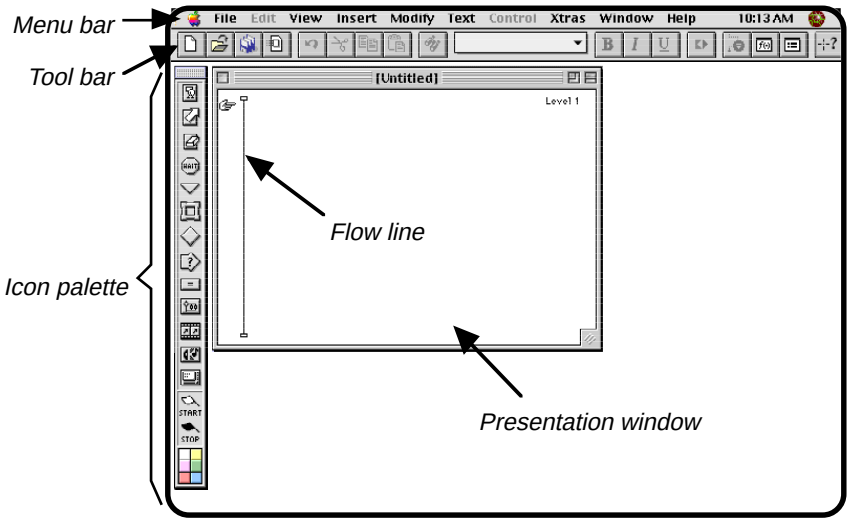
Getting started

The first step is to launch the Authorware application.

☐ **Start Authorware by double-clicking on the Authorware application.**

Say hello to the flow line

Your screen should resemble the following (critical areas are labeled):



For now, we want to focus our attention on the flow line and the icon palette. As described in the preface, your Authorware program consists of a flow chart that you build by dragging icons from the icon palette and dropping them onto the flow line. Don't be shy in putting icons on the flow line. Once there, they can be rearranged, grouped, or deleted.

The flow chart you build here is not only a visual representation of your program, is actually *is* your program.

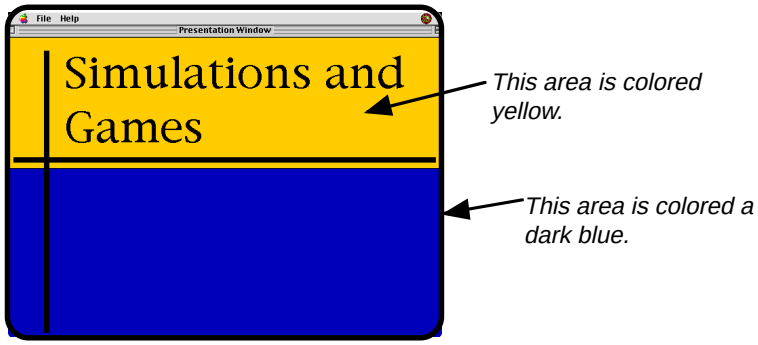
Here is a handy illustration showing the names of all the Authorware icons (the names will give you a good clue to what each does):

Display icon
Motion Icon
Erase Icon
Wait Icon
Navigate icon
Framework icon
Decision icon
Interaction icon
Calculation icon
Map icon
Digital movie icon
Sound icon
Video icon
Start flag
Stop flag
Icon color palette

Of these, only the Decision, Calculation, and Video icons are not used in this chapter. However, the Decision and Calculation icons figure prominently throughout the rest of the book.

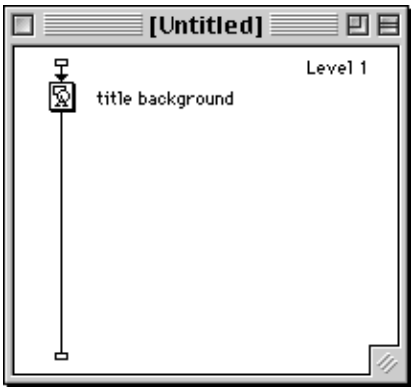
The Video icon controls external video devices and will not be used in this book.

Rather than explain each icon in detail, we are going to just starting using them to build our slide show. For example, our slide show should start with a proper title. Here is what we are after:



You may have already noticed the big empty area at the bottom. Don't worry, we will fill that space up later with a catchy slogan. Let's begin building this display.

☐ Drag a Display icon to the flow line and name it "title background."



There are no hard and fast rules for naming icons. Frankly, it is not even required that you name icons in most cases — you could leave all the icons used in this chapter "untitled" and the program would work just fine (other chapters will show instances where icon names are critical). However, you do yourself a favor when you give an icon a name that informs you of the function it will serve in the program.

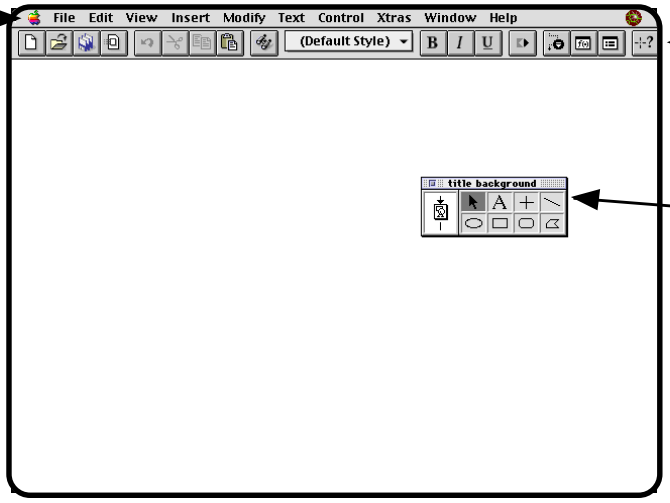
You should be able to name it just by typing. If nothing seems to happen, click *once* on the icon to highlight the text and start typing again.

At this point, try to imagine that the icon is "empty." To tell Authorware what work this icon is supposed to do, we need to open it by double-clicking on it. Some people call this "defining the icon."

Try to remember to click only once when you want to select something. Once selected, it can be cut, copied, or just plain deleted (by pressing the Delete key on the keyboard).

☐ Double-click on the Display icon "title background."

The screen changes dramatically to the following:



You are now "inside" the display.

For all practical purposes, Authorware now becomes an object-oriented (OO) graphics package similar to the many drawing applications on the market. Not surprisingly, in an object-oriented graphics environment you construct and manipulate *objects* on the screen. The computer will retain information about each object, allowing you to easily move, reshape, and reformat each object as often as you like. This is in contrast to "paint" graphics environments in which you paint graphics and "text" on the screen merely changing the

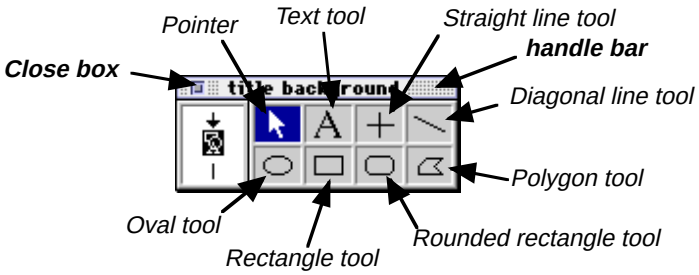
Any experience you have with other object-oriented graphics applications should transfer well to Authorware.

graphic "bit map" of the computer's memory. To modify a painted object, you basically have to "erase" the original and start over. Authorware has *no* paint characteristics. However, it is assumed that you will be using Authorware in tandem with one or more graphics applications of your choice. You can always "copy and paste" graphics from other applications into Authorware — including from paint programs — each then become individual objects once pasted into Authorware. These are concepts more easily demonstrated than described.

The tool box

It pays to get very familiar and comfortable with the tool box from the start. Here are the names of the tools and other critical areas:

Clicking on the close box is how you tell Authorware to "accept" the graphic objects you have placed or modified on the screen and then to go back to the flow line.

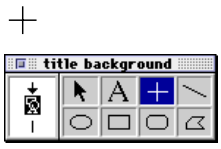


To move the tool box to another spot on the screen, just click and hold on the handle bar, then drag the tool box to the new location.

Like most palettes, you can move this tool box anywhere you want on the screen just clicking and holding on the handle bar, then dragging.

Let's start off slow and easy. Let's begin by drawing two fat dividing lines on the screen. These will help divide up the screen for the members of our audience.

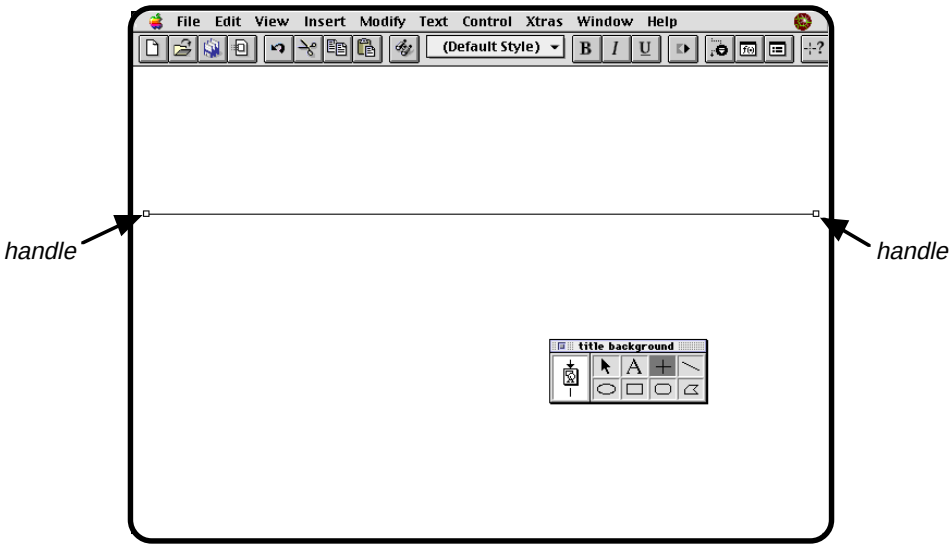
You'll notice that there are two line tools, one for drawing straight lines and the other for drawing diagonal lines. Since our two lines need to be perfectly horizontal and vertical, the straight line tool will be the best choice.



☐ **Click once on the straight line tool.**

The tool icon in the tool box becomes highlighted and the mouse pointer changes into a cross hair.

☐ **Point to the left edge of the screen, about a third of the way down; click and hold the mouse button down while you drag to the right edge of the screen; release the mouse button.**



The first thing you will probably notice is that our line is not very thick, or "fat." Obviously, the default thickness is set to thin. No problem. This is an object-oriented graphics package, so we can just tell Authorware to change this particular characteristic of the line.

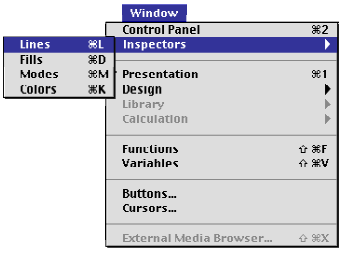
It's very important to note that before we can modify a graphic object, it must first be selected. Notice the two little boxes at either end of the line. These are usually referred to as handles, because these are the critical points at which you can reshape the line. The important point to remember now is that you only see the handles *if* the object is selected. Once selected, the screen object can also be cut, copied, or deleted.

If you don't see the line handles, click once on the pointer in the tool box, then click once on the line.

To unselect an object, click in any "neutral" spot on the screen with the pointer (i.e. a spot where there are no objects).

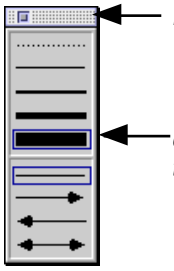
Let's change the width of this selected line from thin to thick.

☐ **Select "Inspectors" from the "Window" menubar, then select "Lines".**



The line palette pops open giving us choices for our lines.

☐ **Click once on the thickest line you see:**



Just like the tool box, you can drag this palette anywhere on the screen by clicking, holding, and dragging it by the handle bar.

If your line was selected, it should now look much thicker than before.

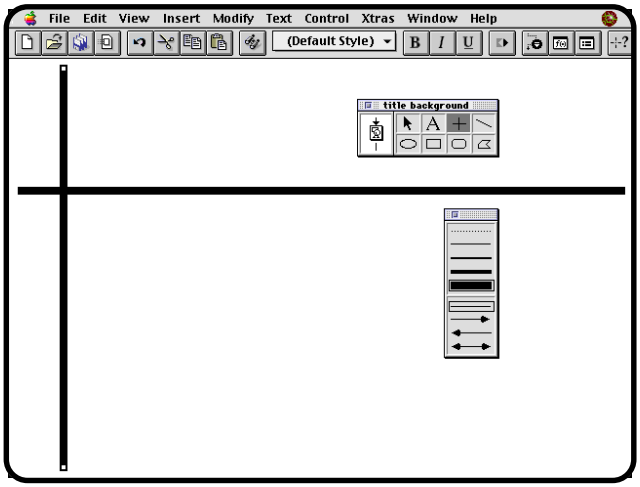
While we are here, notice the other choices you are given. You can give the line an arrow head so that it is pointing in either or both directions. The dotted line at the top means to make the line invisible. This may sound silly at the moment, but there are actually many uses for invisible lines (such as creating a colored box or circle without an edge).

Let's draw the other line going vertically near the left edge of the screen.

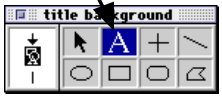
☐ **Click once on the straight line tool** (if not already chosen)**.**

☐ **Point to the left edge of the screen, very near the top; click and hold the mouse button down while you drag to near the bottom edge of the screen; release the mouse button.**

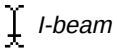
Your screen should resemble the following:



Next, let's add the text of our title.



☐ **Click once on the text tool in the tool box to select it.**

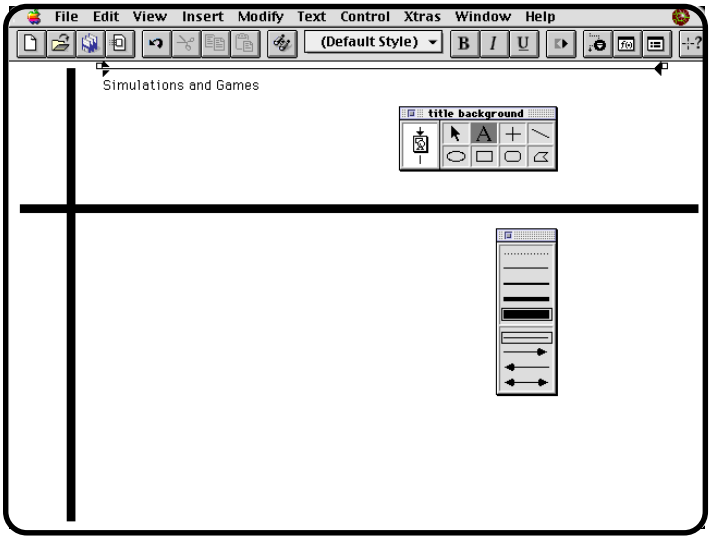


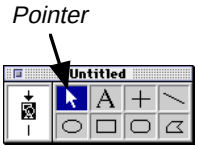
The mouse should now change into an I-beam, the standard symbol for text processing.

☐ **Click once in the top left of the screen.**

☐ **Type "Simulations and Games."**

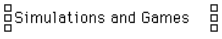
Your screen should resemble the following:





As before, your expectations about the type size and style may not have been met. Again, since this is an object-oriented environment, we can now change this text as we wish.

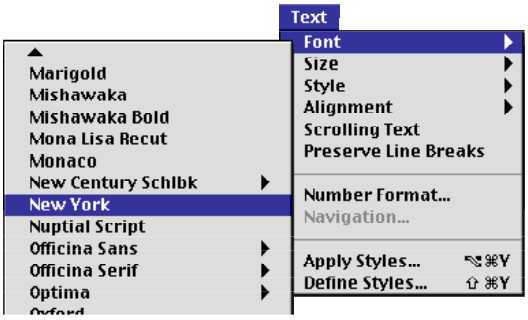
☐ **Click once on the pointer in the tool box.**



☐ **Click once on the words "Simulations and Games" to highlight this text object** (if it is not already selected; handles appear around the words when selected).

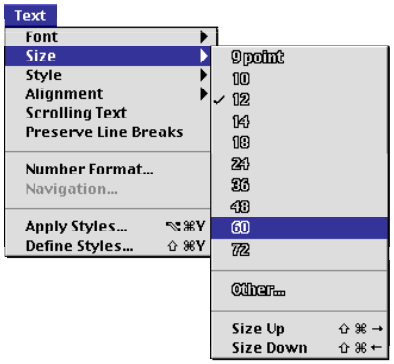
Let's start by changing the font from Geneva (the default font) to New York.

☐ **Select "Font" under "Text" from the menubar, then select "New York".**

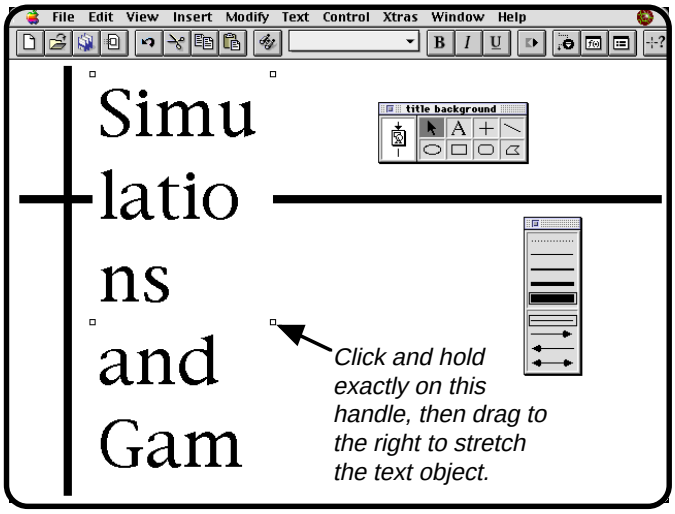


If the text was selected, it should now have changed to the New York font. At this size, the change is subtle, so let's now change the font size to something that an audience of 200 will be able to read.

☐ **Making sure that the text object is still selected, select "Size" under "Text" from the menubar, then select "60".**



Your screen probably resembles something like this:



As you can see, Authorware tried to constrain the text within the original text box. All we need to do now is stretch the text box to the right to make it wider.

An object's handles are critical to reshaping the object. You must click and hold *exactly* on one of the handles as you move the mouse. It is also important to click, hold, and stretch the correct handle. Selecting the correct handle is intuitive. If you are going to stretch the right edge of an object, for example, choose a handle on the right. Handles on the corners of an object allow you to stretch both horizontally and vertically. Handles in the middle of an object's side only allow you to reshape the object along one dimension.

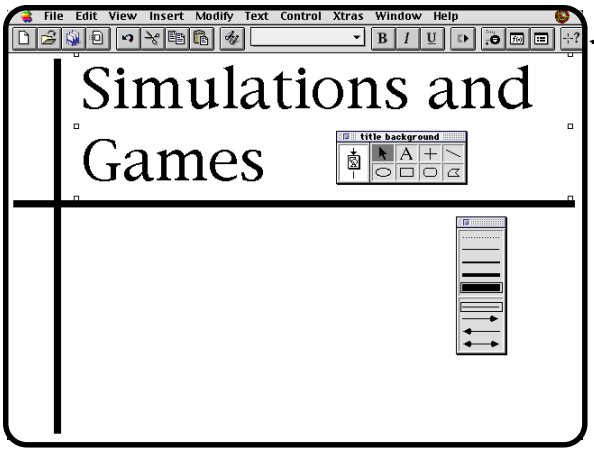
☐ **Click and hold exactly on the handle on the right of the object, then stretch the object to the far right edge of the screen; release the mouse button.**

Undoubtedly, you also want to adjust the position of the text object slightly.

☐ **Click and hold on any one of the letters in the text object, then drag around on the screen to fine tune the text object's position.**

Why click on a letter? This is to ensure that you are grabbing on to something definitely associated with that object and not a "neutral" screen spot.

When done, your screen should resemble the following:



Adding background colors to the display

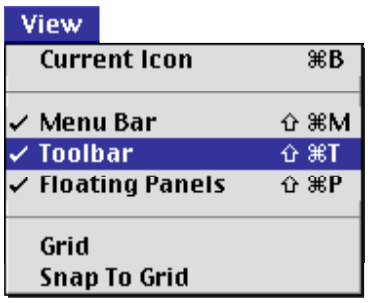
Next, let's add some color to the screen. Let's add a yellow rectangle behind the words "Simulations and Games" and a dark blue rectangle to cover the bottom portion of the screen. One way to do this is to draw a rectangle that fills this part of the screen, then fill the rectangle with a solid blue pattern.

Given that this book is a grayscale product, you will need to use your imagination in all discussions referring to color. (You will find color in the electronic version of the book, though I fudged some of the colors to produce a better grayscale printout.)

Since we will be working near the top of the screen, it is probably a good idea to hide the Tool Bar.

☐ **Hide the Tool Bar by selecting the checked words "ToolBar" under "View."**

The tool bar simply gives you a shortcut to some of the most used tools.

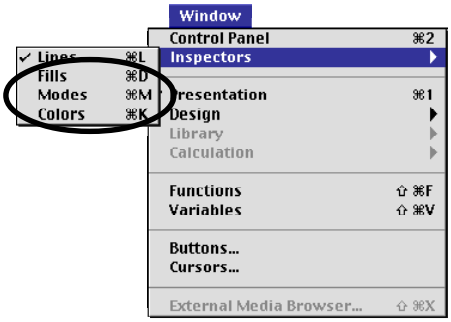


If you accidentally choose to turn off the Menu Bar by mistake, remember that you can toggle it back on by pressing Shift-Command/Control-M.

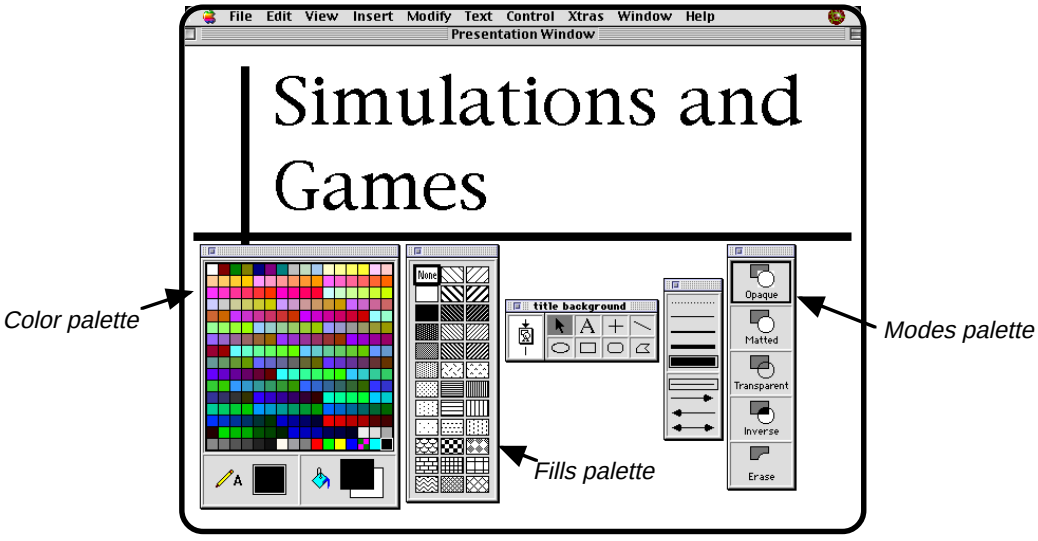
You can toggle the display of this Tool Bar on and off.

It will also be convenient to have three more Attribute palettes open and ready to be used: Fills, Color, and Modes.

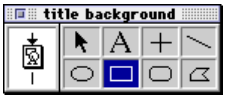
☐ **One by one, select "Fills...", "Modes...", and "Color..." from "Inspectors" under the "Window" menu.**



☐ **Position these three palettes at convenient spots toward the bottom of the screen:**

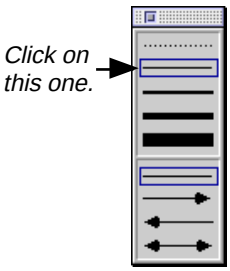


You'll soon see how these palettes work. Now let's draw the rectangle.



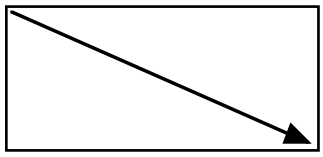
☐ **Choose the rectangle tool in the tool box.**

First, notice that the line thickness is still set to "fat." The rectangle we are about to draw will have an edge set to this thickness. Let's change the line thickness to thin.



☐ **Click on the thin line in the line attributes dialog box.**

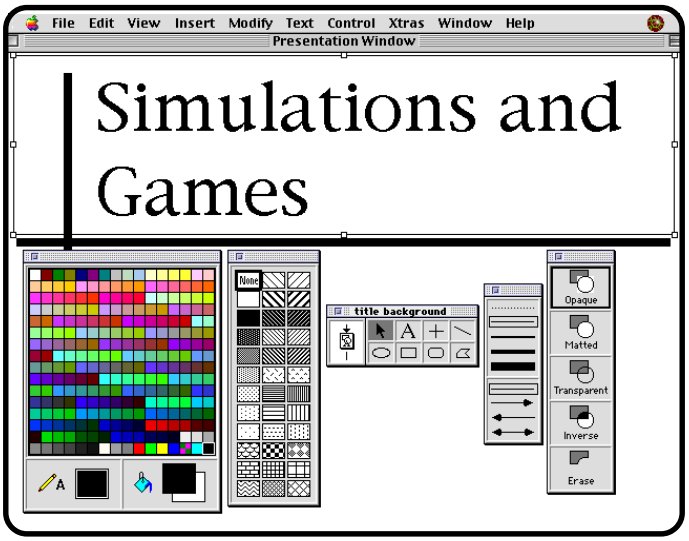
We are finally ready to draw the rectangle. Like all graphics applications, rectangles and ovals are drawn from corner to corner:



To draw a rectangle or oval, click and hold in any one of the four corners, then drag the mouse to the opposite corner.

☐ **Draw a rectangle that fills the screen zone behind the title text by clicking and holding at the top left corner of the screen and dragging to the bottom right corner of this screen zone.**

When finished, you should have something that resembles this:



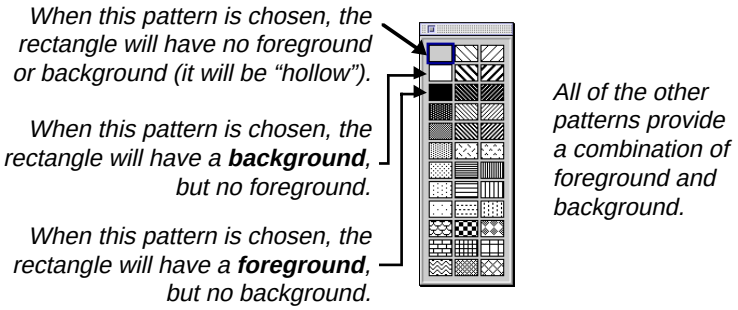
Remember, you can resize the rectangle you have drawn by clicking and dragging one of the handles. To move the entire rectangle *without* reshaping, click and drag one of the edges (be sure to stay clear of the handles).

Filling in this rectangle in with a shade of yellow involves two steps: choosing a pattern from the Fills palette, then choosing a color from the Color palette.

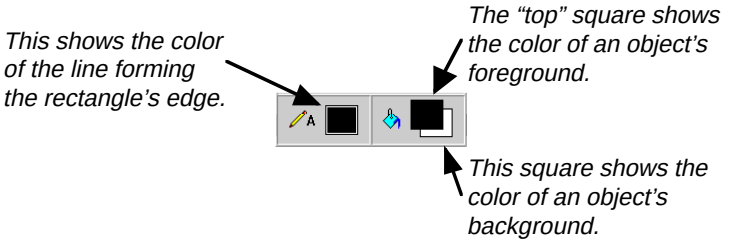
Rectangles have three elements that can have separate colors: foreground, background, and the line that forms the edge.

The first fill pattern has no foreground or background — the rectangle is just a hollow frame. The second fill pattern (white square) fills in the rectangle with a background, but no foreground. The third fill pattern (black square) fills in the rectangle with a foreground, but no background. All of the other fill patterns have a combination of foreground and background.

Don't feel bad if you are confused about this at first. Even experienced Authorware users get disoriented with this every now and then. Understanding will come with experience.

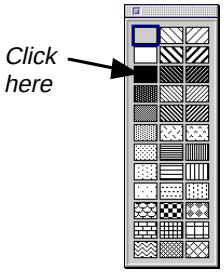


When choosing colors, it is important to know which of these three elements you are changing. At the bottom of the Color palette the color square to the left (beside the "A" and pencil) control the color for the line forming the rectangle's edge. To the right, you see two color squares beside the dripping paint can: the one on top controls the foreground color and the one below controls the background color.



As you change colors, only the chosen element is affected. By default, the foreground color is selected. Click on any of these three color squares to select the element whose color you wish to modify.

To make this rectangle a solid color of pale yellow, we first select the pattern associated with "all foreground, no background" from the Fills palette.

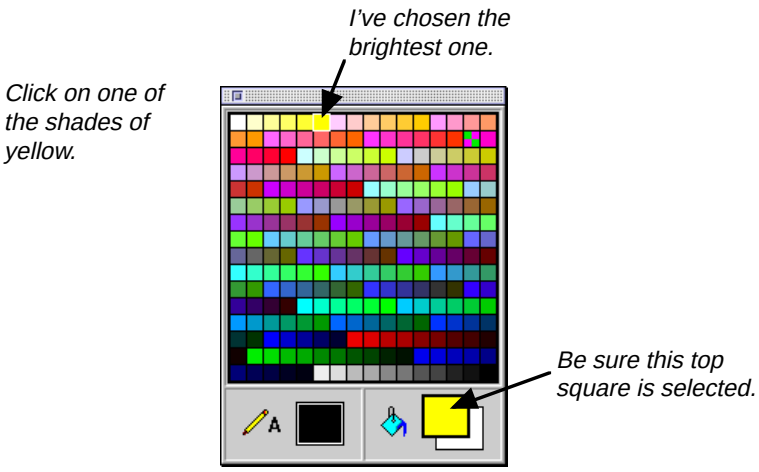


☐ **In the Fills palette, click once on the solid black square (as shown to the left).**

This changes the rectangle to the color chosen for the foreground. Unfortunately, the rectangle now covers the text and one of the lines. We'll take care of that problem later.

Now let's change the color of the rectangle's foreground from black (the default color) to a shade of yellow.

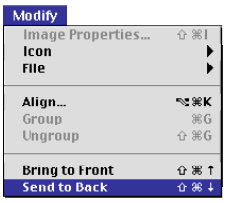
☐ **In the Color palette, be sure that the foreground color square is active, then click on a shade of yellow.**



The rectangle's foreground color should now change to yellow.

Now, let's deal with the fact that the rectangle is covering the text. Like any object-oriented graphics environment, objects are either "on top of" or "behind" other objects. Generally, the objects constructed most recently will be on top. So, we need to tell Authorware to "move" the yellow rectangle "to the back."

Again, be sure that the rectangle is selected (you need to see the handles).



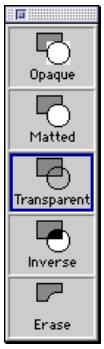
☐ **With the rectangle selected, select "Send to Back" from the "Modify" menu.**

The rectangle will move to the back of all other objects on the screen, thus revealing the text.

Of course, this now uncovers another small problem. The text is shown within a block of white. This is because all text objects, like all objects, also have a foreground and background. The text is the foreground, and the surrounding area of the text object is the background. Although we could change the background color of the text to match the rectangle's color, a better strategy is to change the "mode" of the text object from "opaque" to "transparent." This will, in effect, tell Authorware to "remove" the text object's background letting whatever is beneath to show through.

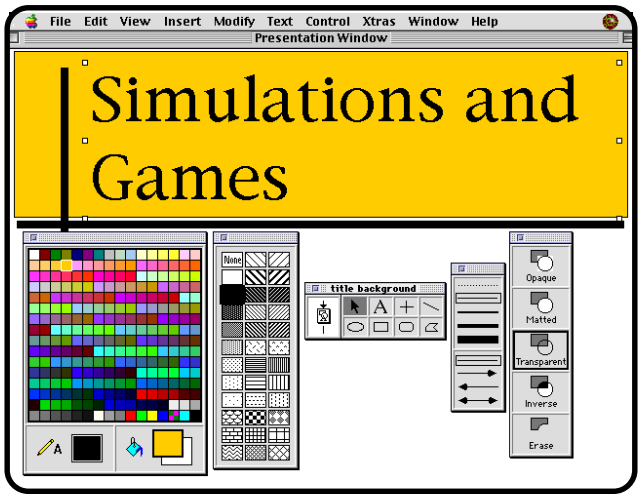
☐ **Select the pointer from the tool box** (if it is not already chosen).

☐ **Click once on the text object to select it (be sure handles appear around the text object confirming it has been selected).**



☐ **In the Modes palette, choose “Transparent.”**

The black text should now blend nicely on the yellow rectangle:



Now let’s add a blue rectangle to the bottom of this screen. The procedure is the same as the first rectangle.

☐ **Choose the rectangle tool in the tool box.**

☐ **Click on the thin line in the line attributes dialog box** (if it is not already chosen)**.**

☐ **Draw a rectangle that fills the screen zone below the horizontal thick line by clicking and holding at the middle left of the screen and dragging to the bottom right corner of the screen.**

When finished, you should have something that resembles this:

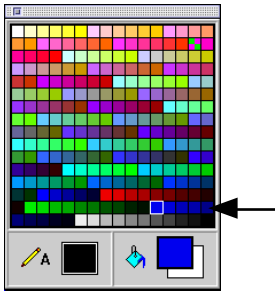
This second rectangle is also yellow because the defaults for the fill and color palettes automatically change to the last fill and color chosen.

Remember, you can resize the rectangle you have drawn by clicking and dragging one of the handles. To move the entire rectangle without reshaping, click and drag one of the edges and stay clear of the handles.

Next, we change the foreground color from yellow to blue.

☐ **In the Color palette, be sure that the foreground color square is active, then click on a shade of blue.**

The rectangle's foreground color should now change to blue.



☐ **With the rectangle selected, select "Send to Back" from the "Modify" menu.**

Fine-tuning the position of the screen objects

The last step in constructing this display is the minor adjustment of all the screen objects. In particular, we want the yellow and blue rectangles to fill the screen. We also want them to touch right behind the horizontal thick line. To do this, we simply select the pointer from the tool box, then alternately select and adjust the position of each rectangle.

☐ **Select the pointer in the tool box.**

☐ **Fine tune the position of each rectangle so that the screen resembles that shown on the next page:**

☐ **If necessary, fine tune the position of the thick lines and text.**

We are done! To close this display and return to the flow line, click in the close box in the upper left hand corner of the tool box.

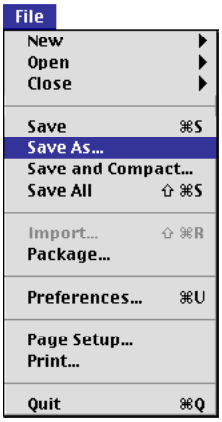
☐ **Click in the close box of the tool box to return to the flow line.**

You should now be back at the flow line:

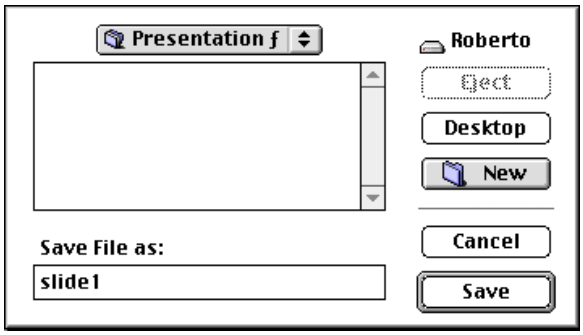
Congratulations! You just constructed your first Authorware display.

Saving the file

Let's save the file. Like working in any application, it's a good idea to save early and often.



☐ **Select "Save as..." from the "File" menu.**



☐ **Save your file with the name "slide1" to the folder of your choice.**

From here on, remember to save your file periodically using the "Save" command from the "File" menu. (Only use "Save as..." when you want to save another copy with a different name or to a different disk location.)

After you save, you will notice that the extension ".a4p" will automatically be added to your file name. This can be deleted later if you want.

We'll keep to the "8 dot 3" rule for naming files, even though most users do not have this limitation.

Finishing the title sequence

To finish the title sequence, we are going to add the following subtitle to the blue area at the bottom of the screen: "Powerful strategies for design...powerful interactions for learning." For impact, we will display each of the subtitle's two parts separately. That is, we want to display "Powerful strategies for design..." in the blue area, wait, then erase and replace it with "...powerful interactions for learning." Finally, we will add a simple animation of a space shuttle flying across the screen along with a sound effect.

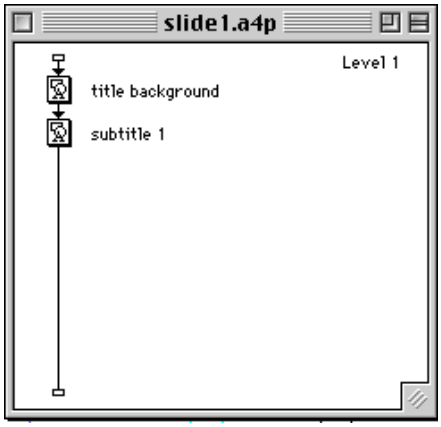
The space shuttle will symbolize the creative "journeys" one can experience while learning with simulations and games.

Adding the subtitle

Displaying the subtitle in two parts smoothly on the title background title we just created requires the use of two separate Display icons for the subtitle. The reason has to do with the way displays are erased in Authorware. In short, it is not possible to erase *part* of a Display icon — it's an all or nothing proposition. This will become clear in a few minutes. Let's begin.

☐ **Drag a Display icon to the flow line and drop it just below the Display icon titled "title background."**

☐ **Title this new Display icon “subtitle 1.”**

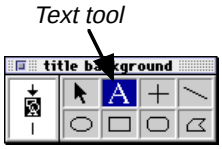


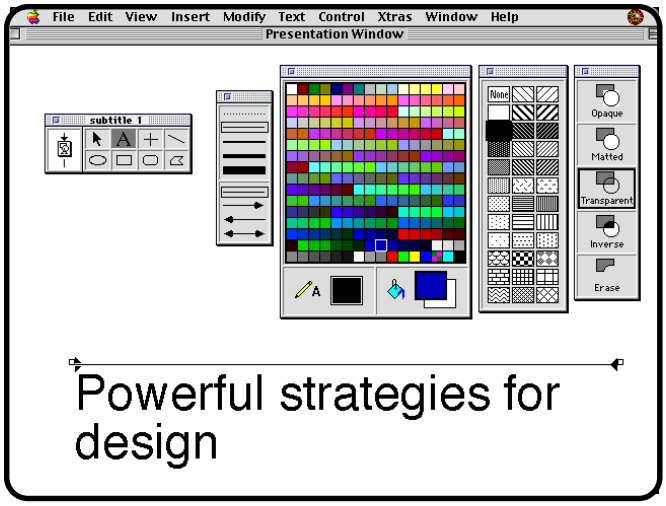
As before, we’ll “define” this icon by double-clicking on it.

☐ **Double-click on the Display icon “subtitle 1” to open it.**

Here’s a good chance to apply what you just learned.

☐ **With the text tool, type the following on the screen using Helvetica font with a size of 48 point:**





Move the various palettes out of the way while you type. Of course, you could always close these palettes (by clicking in the close box in the top left hand corner of each) and open them again later when needed. You can also double-click on a palette’s handle to activate the “window shade” feature.

Don’t worry too much about the exact placement of this text on the screen. As you’ll soon see, it is very easy to fine tune the position later.

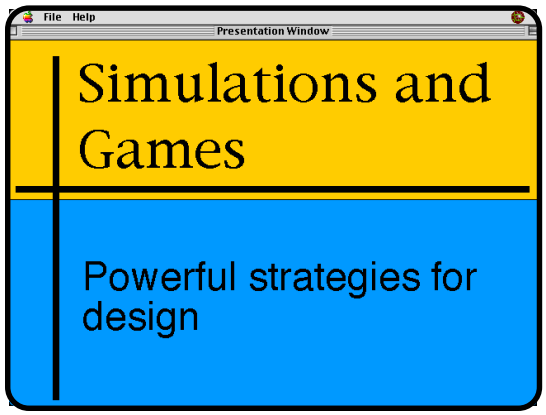
☐ **When finished, click in the close box in the tool box to go back to the flow line.**

Running the program

We are finally ready to run our slide show for the first time. Running, or restarting, the program is the same as asking Authorware to execute the flow line, starting at the top and then following the flow line down through the program. Therefore, the program should start by displaying the title background followed by displaying the first subtitle. Let's try it.

☐ **Select "Restart" from the "Control" menu.**

Authorware executes your flow line. The "background title" is displayed followed by the "subtitle 1" display, at which point the program ends. Your screen should resemble the following:



The graphic design of these displays obviously needs some work. First of all, the black letters on the dark blue background are very difficult to read. We will change the text color to yellow (this will offer a nice contrast to the text and background colors in the area above). Second, the subtitle needs to be spaced better on the screen. Third, you may also want to adjust the position of the objects in the title background (e.g. the lines, text and two background colors). No problem. In fact, it is assumed in Authorware that you really won't know how things will look together until you finally run the program. Therefore, Authorware gives you many easy ways to modify and revise your displays after they have been initially constructed. The simplest is that you can quickly move the contents of an entire display just by clicking, holding, and dragging the display *while running your program.* Let's try it.

Don't worry, other users won't be able to do this after the file is packaged — unless you want them to: check out "Effects" under "Attributes" the next time you open a Display icon.

☐ **Fine tune the position of the subtitle by clicking and holding on any of the subtitle's letters, then dragging the subtitle to a new location.**

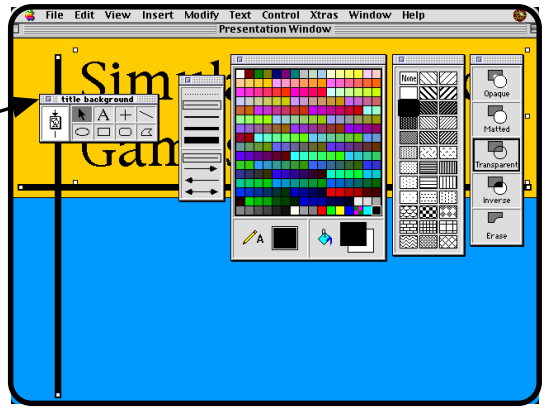
However, this technique won't work if you want to fine tune the position of just the words "Simulations and Games." If you try to move these words you'll find that all of the objects associated with this display move as well. No problem. Just

Reminder: This little trick only allows you to move <u>all</u> the contents of a Display icon.

double-click on one of the objects in this display. This will tell Authorware to reopen that Display icon. Let's try it.

☐ **Double-click on the words "Simulations and Games" to reopen the Display icon "title background":**

Always make sure that you have opened the correct Display icon by confirming that its title appears here.



Notice that the tool box and all palettes that were left on the screen reappear. Also notice that the title of the Display icon is in the tool box (this is useful information in case you accidentally double-clicked on an object in *another* display). Also notice that there are handles around every object. In other words, every object is selected. Let's fine tune the position of just the words "Simulations and Games."

☐ **Click once on the words "Simulations and Games."**

The handles around the other objects should disappear.

☐ **Fine tune the position of this text object.**

☐ **Fine tune the position, size, or shape of any other object in this display.**

Here is the final result of my modifications:

I obviously closed all of my palettes.

If you have any white area of the screen showing, you can make the rectangle "oversized" and then drag it so that it overlaps the edge of the screen a little bit.

Clicking here makes a line invisible.

The only tricky part was that I made the lines defining the edges of the two colored rectangles invisible and then *very carefully* aligned them together on the screen.

Notice that I added a copyright notice at the bottom right. Don't be shy in protecting your copyright to works you create. See Appendix F for more information.

☐ **When finished, click in the close box of the tool box.**

This sends us back to where we were just before we double-clicked on an object in the display — back to the running of the program. That's convenient because now we can just double-click on the subtitle to open its display and make some changes.

☐ **Double-click on one of the letters in the subtitle.**

☐ **Make sure that the Display icon title "subtitle 1" appears in the handle bar of the tool box.**

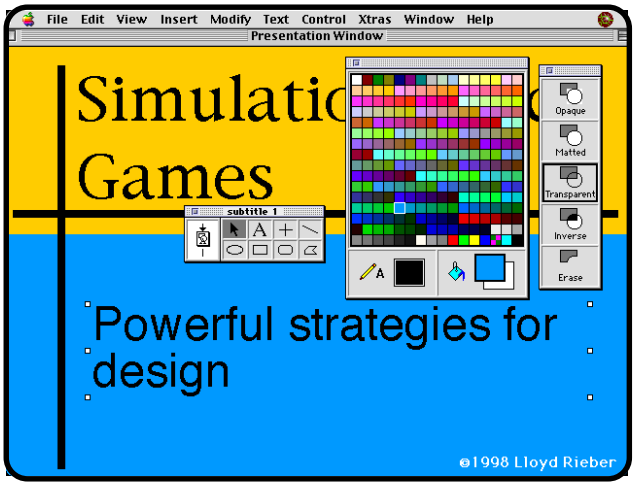
I have found that many novices get confused about this. Remember, Authorware can't read your mind. All it knows is the exact spot at which you double-click (the tip of the pointer's arrow).

If it doesn't, then you accidentally double-clicked on an object in the other display. If this happened, click on the close box in the tool box and try double-clicking on the subtitle again — aim right on one of the letters.

This text object needs to have the text color set to yellow. (Its mode also needs to be set to transparent if it is not already so.)

☐ **Be sure the text object is highlighted (handles appear) and that the color palette and mode palette are open on the screen.**

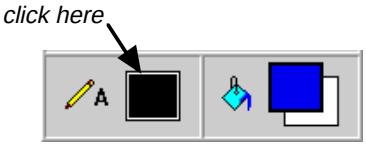
Reminder: You open these palettes by selecting them from "Inspectors" under the "Window" menu.



If you are feeling daring, try out some of the special effects associated with all Display icons. Select "Icon" from the "Modify" menu, then select "Properties..." and explore.

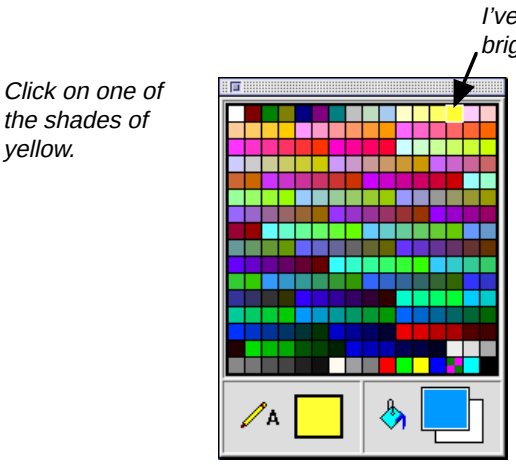
☐ **Be sure mode is set to transparent.**

☐ **In the Color palette, click once on the color square on the left (beside the pencil and "A") to make the text color active:**



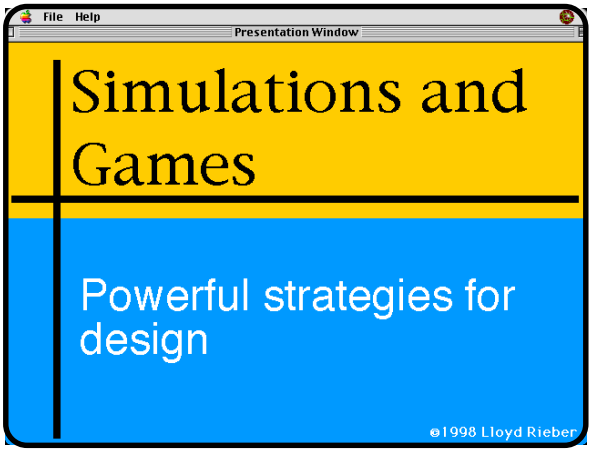
Now we can change the color of the text.

☐ **In the Color palette, click on a shade of yellow.**



☐ **When finished, click in the close box in the tool box.**

Your screen should resemble the following:



The idea is for the shade of yellow of the background above to match the shade of yellow of the letters below. (A reminder to those viewing the electronic version of the book that I am fudging the colors a bit to optimize the grayscale print-out.)

We now want to return to the flow line. But how? You may recall seeing the command "Stop" under the "Control" menubar. Unfortunately, this "Author Menu" does not appear when running the program. There are a couple of ways to do this, one of the simplest being just choose "Quit" under the "File" menu. When you are running the file, Authorware switches to a "User Menu" and assumes that "Quit" here means just to quit running the program and to go back to the flow line. But if you choose "Quit" when you are already at the flow line, this means to quit the Authorware application. This risks not only an inconvenient loss of time, but the potential to accidentally quit without saving. Because of the tendency of new users to get confused about whether "Quit" really means "Quit," we will opt to use the keyboard shortcut "Command/Control-J" — with "J" standing for "jump back to icons." Interestingly, unlike earlier versions of Authorware you will not find a menubar option to correspond with "Command/Control-J" in Authorware 4.0. We will use "Command/Control-J" as the main way to return to the flow line throughout this book.

"Quit" will really mean quit when the file is running only after it is packaged, a process described in Appendix A.

I'm guessing that this is either an oversight in Authorware 4.0, or the decision was made to keep this keyboard command because it is second nature to the fingers of thousands of long-time Authorware programmers.

Command Windows

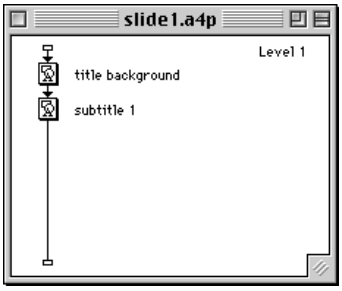
☐ **Press "Command/Control-J" to return to the flow line.**

Platform Alert!

Macintosh users use the Command key and Windows users use the Control key for all keyboard shortcuts. (No further reminders will be given in this chapter.)

Setting up the second subtitle

You should now be back at the flow line:

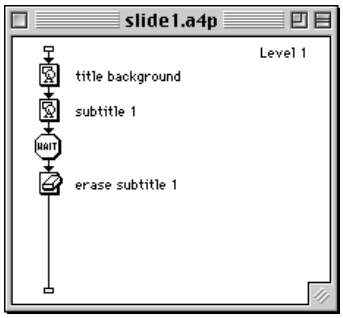


After the "title background" and "subtitle 1" are displayed, we want the program to pause until the user presses a key or mouse button after which "subtitle 1" is erased, followed by "subtitle 2" taking its place. At that point, the program should pause again. After the user presses a key or mouse button, "subtitle 2" and the "title background" will be erased.

☐ **Drag a Wait icon to the flow line and deposit it below "subtitle 1."** (Wait icons are usually not titled.)

☐ **Drag an Erase icon to the flow line and deposit it below the Wait icon; title it "erase subtitle 1."**

Your flow line should now look like this:

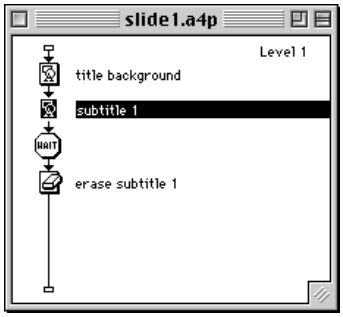


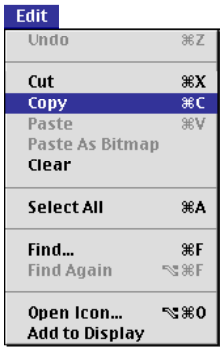
In a moment, you'll see how to tell Authorware to erase a display with the Erase icon. But let's continue designing this sequence. The next icon we need to construct is a Display icon for "subtitle 2." Although we could construct this from scratch, we will instead take the convenient shortcut of "copying and pasting" the Display icon "subtitle 1." We can then just change the text while keeping all of the formatting intact.

Learn these shortcuts. They can save you much time and effort.

Before you can copy something, you have to first select it.

☐ **Click once on the Display icon "subtitle 1":**



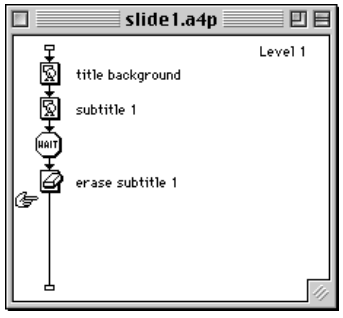


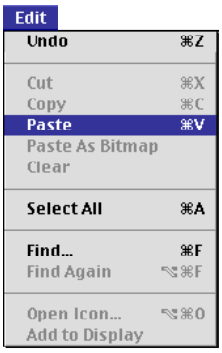
☐ **Select "Copy" from the "Edit" menu.**

This copies the Display icon "subtitle 1" (and all its contents) to the clipboard.

☐ **Click once on the flow line just below the Erase icon "erase subtitle 1" so that the "paste hand" appears:**

paste hand





☐ **Choose "Paste" from the "Edit" menu.**

Icons previously cut or copied are pasted to wherever the paste hand is pointing.

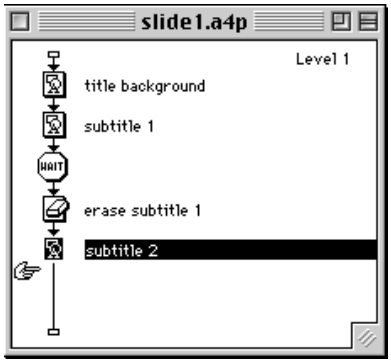
You now have another Display icon titled "subtitle 1." Let's rename it "subtitle 2."

☐ **Click once on the Display icon just pasted.**

The title of the icon should now be highlighted.

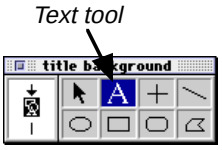
☐ **Rename this icon "subtitle 2"** (just by typing)**:**

You can also just click once at the end of the title, delete the last character, then type a "2." Icon titles act as 1-line text windows.



Now all we have to do is double-click on this Display icon to open it, then change the text to the second part of the subtitle: "...powerful interactions for learning."

☐ **Double-click on the Display icon "subtitle 2."**



☐ **Select the text tool from the tool box.**

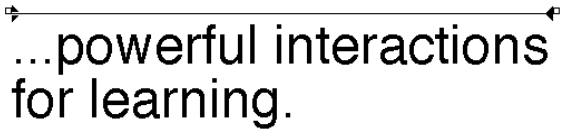
☐ **Click inside the text object to activate it.**

☐ **Click and hold just before the first letter of this text phrase, then drag to the end of the phrase to highlight the entire phrase.**

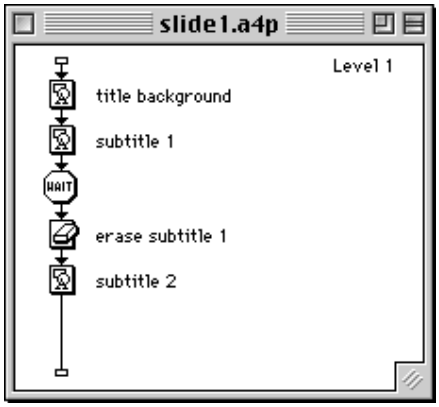
☐ **Type "...powerful interactions for learning."**

The yellow text may be hard to see on the white background (recall that the blue background is part of the Display icon "title background").

The text size, style, color, mode, and location will remain intact — only the text itself changes:



☐ **When done, click in the close box in the tool box to go back to the flow line:**



We are ready to run our program again. Again, running the program simply means that Authorware will execute the icons on the flow line in the order in which they appear, starting at the top and working down.

However, you may be wondering about that Erase icon — how does Authorware know which icon to erase? There are a couple of ways to do this. We will use a method that demonstrates a wonderful feature of Authorware. We are merely going to run the program. Anytime Authorware encounters an icon on the flow line that has not yet been defined, it will automatically stop the execution of the program and open up the icon (an Erase icon in our case) for us to deal with. You'll understand this much better just by doing it.

☐ **Run your program by selecting "Restart" under "Control" (or by pressing "Command/Control-R").**

The first thing you see is the subtitle superimposed on the title background. Also notice that a "Continue" button has magically appeared. This was generated automatically by the Wait icon.

Authorware put this continue button in a strange place! Not to worry — we can easily move or delete it later.

☐ **Click on the "Continue" button to continue.**

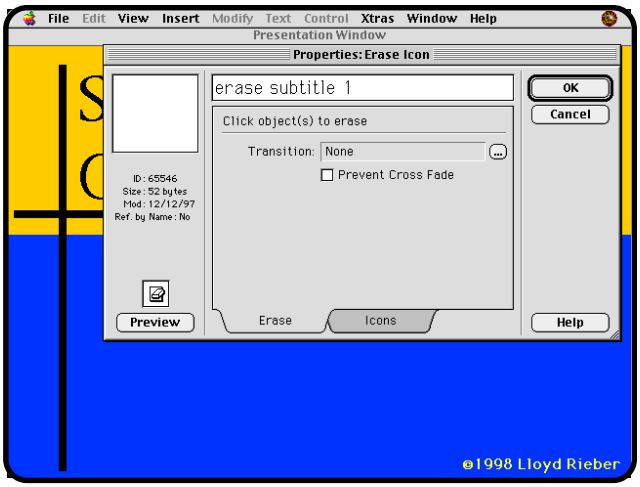
Recall what icon comes next on the flow line — the Erase icon titled "erase subtitle 1." Since this has not yet been defined, Authorware interrupts the execution of the program and displays this icon's dialog box.

Notice that this dialog box also has two parts, as indicated by the tabs "Erase" and "Icons".

What to do next? Well, the directions are right there in the dialog box. Simply click on the display you wish to erase. An Erase icon can only erase the *entire* contents of a Display icon. Fortunately, what we want to erase — the text "Powerful strategies for design... — is the only thing in the Display icon "subtitle 1."

Now you can understand why we put this text object in its own display.

☐ **Click once on the text "Powerful strategies for design..." to erase the entire contents of the Display icon "subtitle 1."**



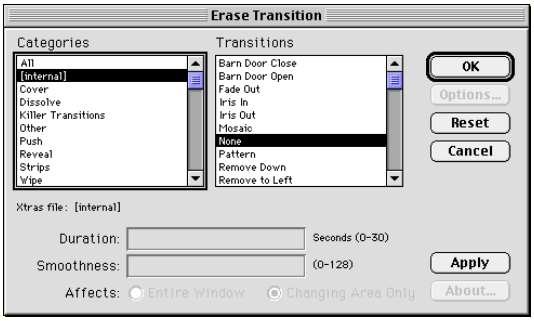
While here, notice that you can choose to erase displays with special effects called transitions.

When used well, these transitions can help establish continuity and a sense of place (such as by using the same transition at particular junctures in a program). If overused, transitions can distract, annoy, and frustrate users. My advice is to be judicious in the use of any "glitzy" effects, such as these transitions, otherwise, your program may resemble a rock video more than instruction.

Click on this symbol:

☐ **Click on the button besides the "Transition" box.**

A dialog box with a variety of transitions appears:



☐ **Test several transition effects by clicking on a transition, followed by clicking on the "Apply" button.**

Feel free to try some of the transitions in the other categories.

☐ **After finding a suitable transition, click "OK."**

I chose the transition "Mosaic" in the "Internal" category.

Feel free to explore these transition effects as you are learning how to design with Authorware.

You should be back at the main dialog box for the Erase icon.

☐ **Click "OK."**

Authorware is now "programmed" to erase this Display icon with this Erase icon using the transition you selected..

The program resumes execution. Subtitle 1 should now be replaced with subtitle 2. Since that is the last icon on the flow line, the program ends (and hangs). Let's run it again.

While here, notice that you can choose to erase displays with special effects. Feel free to explore these transition effects.

☐ **Press "Command/Control-R" to run the program again; go through the program until the end.**

The transition from subtitle 1 to subtitle 2 should be very smooth. Run it a few more times if you wish.

Let's go back to the flow line.

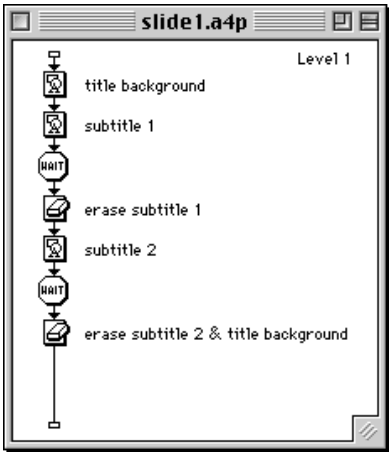
☐ **Press "Command/Control-J" to go back to the flow line.**

Let's finish this sequence by adding another Wait icon and Erase icon to the flow line. After the second subtitle is displayed, we again want to give the user the opportunity to read the screen, pressing the "Continue" button when finished. Then, *both* the second subtitle and the title background need to be erased. We can erase as many displays as we wish with a single Erase icon.

☐ **At the flow line, add another Wait icon and Erase icon to the flow line.**

☐ **Title the Erase icon "erase subtitle 2 & title background."**

I like naming icons with a phrase that tells me what to do. This is a helpful strategy, especially when your programs start getting more complex.



We will again tell Authorware which icons to erase just by running the program and letting it automatically interrupt the execution when it encounters the last Erase icon.

☐ **Run the program (by pressing "Command/ Control-R") and go through the entire program.**

The Erase icon will pop open when you reach the end:

Again, just follow the directions in the dialog box.

☐ **Click once on the text "...powerful interactions for learning." to erase the Display icon "subtitle 2."**

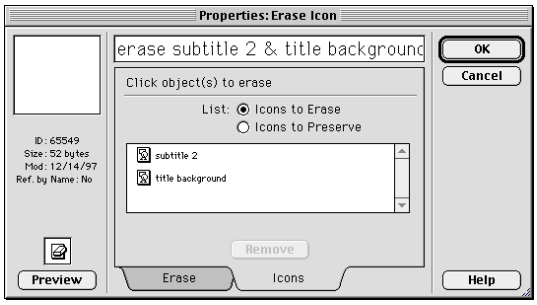
Any object will do because each is associated with this Display icon.

☐ **Click once on any other object to erase the Display icon "title background."**

Feel free to also choose a transition at this point if you like.

Before we click "OK" to accept our programming of this Erase icon, let's take a look at what options or information are available when the "Icons" tab is pressed.

☐ **Click on the tab marked "Icons".**



Notice that the names of the two Display icons we clicked on a few minutes ago appear in the list of "Icons to Erase." If you changed your mind about erasing a display, you could click on its name from the list, then click "Remove."

If you changed the option to list "Icons to Preserve," then all icons *except* the ones listed would be erased. This is a handy shortcut when you want to erase *all* icons currently displayed at some point in a program. We will not make any changes to these options.

☐ **Click "OK."**

As before, clicking "OK" tells Authorware to accept our programming of this Erase icon.

☐ **Restart your program a couple of times to make sure it works properly (by pressing "Command/Control-R").**

☐ **When finished, press "Command/Control-J" to go back to the flow line.**

Have you saved lately?

☐ **Select "Save" from the "File" menu (or just press "Command/Control-S").**

Modifying the Wait icon

Let's make one quick modification. As you know, the Wait icon automatically generates the "Continue" button and puts it at a location not of our choosing. We have several choices here: 1) change the word "Continue;" 2) move this button to another location on the screen; or 3) delete it and allow the pressing of the mouse button or any key advance the program. We will choose the third option at this point for the simple reason that less clutter on the screen is better for a slide show. Of course, this means that the user must know to press the mouse button or any key when the program pauses. This is appropriate since we are, in a sense, just designing this presentation for ourselves. However, designing an interface for another user to understand how to navigate through the program is a very important and involves a complex set of design issues.

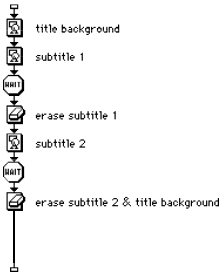
Incidentally, here are brief instructions on how to accomplish options 1 and 2 just in case you want to pursue either of these two options instead.

Many authors get around this limitation by choosing to turn the display of this button off and instead displaying their own button graphics within Display icons throughout the program.

Option 1: To change the word "Continue" to another word or phrase, you have to select "File" from the "Modify" menu, then select "Properties...". You can make the change in the Interaction set of options. Unfortunately, this button will display one and only one word or phrase for the *entire* program.

Option 2: To move (and resize) the button, just run the program, pause the program (by pressing "Command/Control-P") when you see a "Continue" button, move or resize it, then press "Command/Control-P" again to have the program proceed.

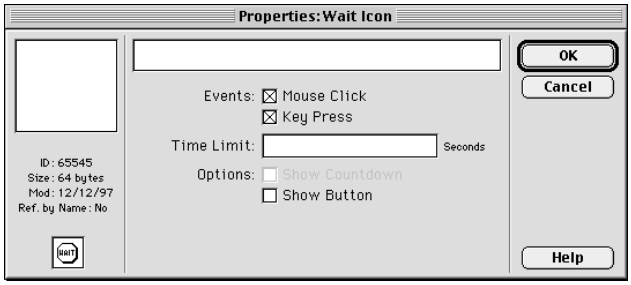
Here are full instructions for the third option — to tell Authorware not to display the "Continue" button and instead just accept a mouse click or key press to step through the title sequence.



☐ **At the flow line, double-click on the first Wait icon.**

☐ **In the dialog box that appears, click once in the select box beside "Mouse Click" so that an X appears** (this turns on this function)**.**

☐ **Click once in the select box beside "Show Button" so that the X disappears** (this turn off this function)**.**



☐ **Click "OK."**

☐ **Double-click on the other Wait icon and repeat the same procedure.**

☐ **Press "Command/Control-R" to restart the program.**

As you step through it this time, you can just click the mouse button (no matter where the arrow is pointing) or press a key.

☐ **When done, press "Command/Control-J" to jump back to the flow line.**

Adding animation and sound to the title sequence

The last thing we will add to this title sequence is a simple animation sequence of a space shuttle flying across the screen accompanied by an attention-gaining sound. This will add a little "razzle dazzle" to the start of our presentation (without getting too carried away).

Actually, the hard part is finding (or constructing) the right graphic and audio segment. For that reason, you may want to just use the resources that came with this book. The resources include some simple clip art saved in the PICT graphics format. You should be able to read this file with any standard graphics application that has "paint" features (such as ClarisWorks or PhotoShop). Again, it is assumed that you are comfortable using a graphics application and know how to "copy and paste" objects from one file to another.

It's a good idea to open the clip art file right now.

☐ **Open the PICT file "Rieber Clip Art" with a graphics application of your choice.**

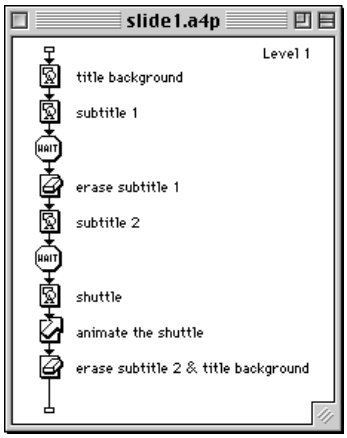
If you don't have access to this clip art file right now and don't want to stop to try to find it, you could just create a graphic "placeholder" for the space shuttle. For example, you could just draw a simple rectangle or circle in place of the shuttle and work on the animation. You can then easily replace this graphic later.

Let's begin by adding another Display icon to the flow line that will hold *only* the graphic of the shuttle. The reason for this is that the Motion icon works in a way similar to the Erase icon — it's an all or nothing proposition. You can only animate the entire contents of a Display icon. When you want to animate just one graphic, you need to embed it in its own Display icon.

☐ **Drag a Display icon to the flow line and drop it between the Wait icon and the Erase icon titled "erase subtitle 2 & title background"; title it "shuttle."**

☐ **Drag a Motion icon to the flow line and drop it just below the Display icon "shuttle"; title it "animate the shuttle."**

Your screen should resemble the following:



We can again use the trick of just running the program and letting Authorware automatically open up both icons for us to define when they are encountered.

Notice that we placed these two icons before the final Erase icon. Why not after? Think about it. We want the shuttle to streak across the screen before we erase the title background and the second subtitle.

Of course, feel free to try other approaches. For example, if you put the second Wait icon immediately after the Motion icon "animate the shuttle" the shuttle will streak across as soon as the second subtitle is displayed.

- ☐ **Press "Command/Control-R" to run the program; step through the title sequence until the Display icon "shuttle" automatically pops open.**

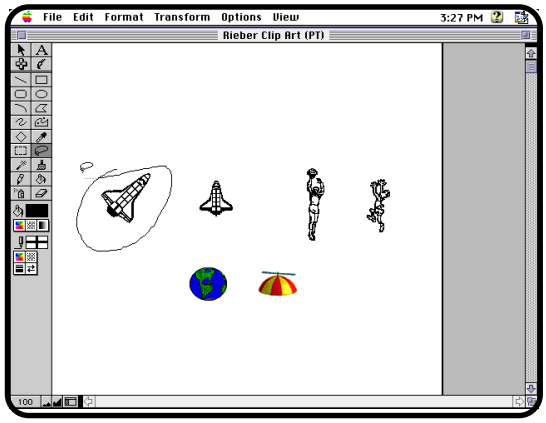
- ☐ **Copy the shuttle graphic from the clip art file and paste it into the open Display icon in the Authorware file.**

Macintosh Alert!

It's expected that you already know how to do this kind of multitasking (i.e. switching between Authorware and a graphics application). For example, here is a snapshot of the clip art file opened with the painting package that comes with ClarisWorks. The shuttle graphic has just been selected with the lasso tool:

Windows Alert!

One way to accomplish this is by pressing "Alt-Tab" to switch between open applications.



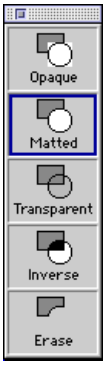
After selecting "Copy" from the "Edit" menu, you would switch back to Authorware and paste it into the open Display icon.

☐ **Click, hold, and drag the shuttle to the bottom left corner of the screen:**

This will make it seem as though the shuttle is doing a "fly by."

This is the "starting point" of the animation.

If the shuttle's white graphic background shows through, this can be fixed by changing the mode from Opaque to Matted.

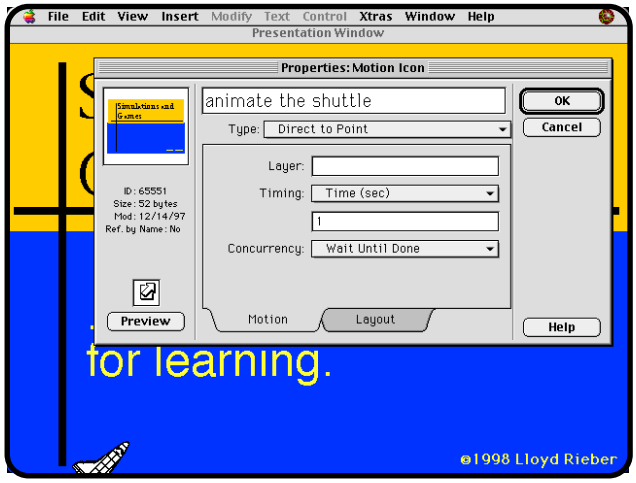


☐ **Be sure the shuttle graphic is highlighted (handles appear), then choose Matted from the Mode palette.**

For an added effect, drag the shuttle so that only half of it appears on the screen:

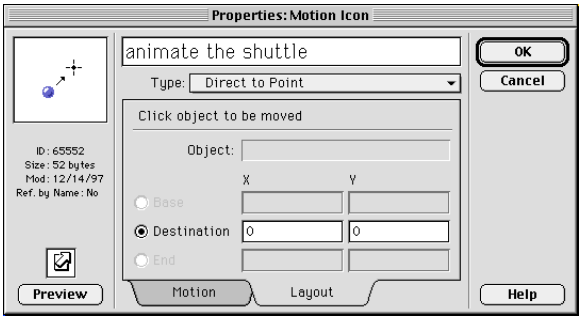
☐ **When done, click in the close box in the tool box.**

Authorware then resumes the execution of the program. The next icon on the flow line is the Motion icon. This also has yet to be defined, therefore, Authorware again automatically pauses and opens this icon for us.



Just like the Erase icon, the Motion icon is divided into two main parts, as indicated by the tabs: Motion and Layout.

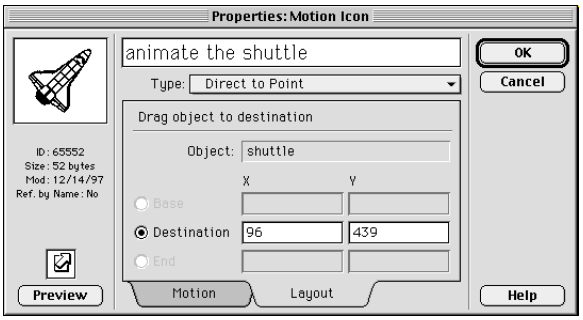
☐ **Click once on the "Layout" tab.**



The instructions of what to do next are contained concisely at the top of the Layout window: "Click object to be moved." Let's do it.

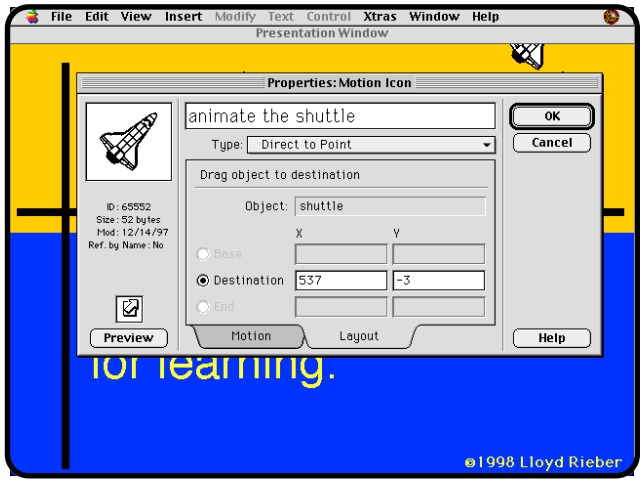
☐ **Click once on the graphic of the space shuttle.**

The dialog box changes to show the display chosen:



Also notice that the directions have changed to "Drag object to destination." Let's do it.

☐ **Click and hold on the shuttle graphic, then drag it toward the top right hand corner of the screen.**



You can test your animation by pressing the button marked "Preview."

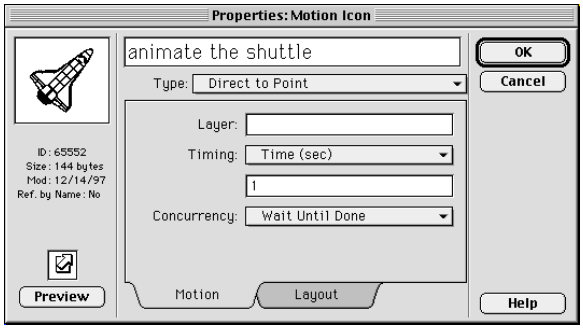
☐ **Click on the button "Preview."**

The shuttle moves across the screen to the exact point where you dragged it in exactly one second. We can edit the animation to change either the final destination or the duration of the animation.

☐ **Fine tune your animation by moving "ending point" of your shuttle's animation.**

To change the timing of the animation, we have to first switch over to the Motion attributes.

☐ **Click once on the "Motion" tab.**



☐ **Enter the number of seconds you wish the animation to take (click "Preview" after each try to test the animation).**

I settled on a final rate of 2 seconds.

Macintosh Alert!

MacOS 8 users do not need to master "click and hold" because of the "sticky menu" feature of MacOS 8.

While here, notice your other options. If you click and hold on "Time (sec), you can choose to have the animation be based on a predetermined "Rate":

Rate is the same as setting a "speed" for your animation. Instead of going so many miles per hour, objects go so many inches per second.

Also notice the phrase "Wait until done." If you click and hold on this phrase, you will see the other option — "concurrent":

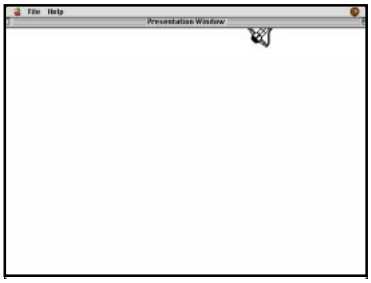
"Wait Until Done" means that Authorware will wait until the animation has finished completely before going to the next icon on the flow line, where as concurrent means that as soon as the animation begins, the next icon on the flow line is executed. The option "wait until done" is appropriate in this case, but this decision (and distinction) will return again shortly when we insert some sound.

☐ **When done, click "OK."**

☐ **Run your file to test it** (by pressing "Command/Control-R".)

The animation should smoothly blend in with the other segments of the title. One problem remains: the shuttle does not get erased. We can either drag another Erase icon to the flow line to handle this, or we can give one of the Erase icons already on the flow line this job. We'll choose the latter.

☐ **Run your file one more time to the end so that only the shuttle remains on the screen:**



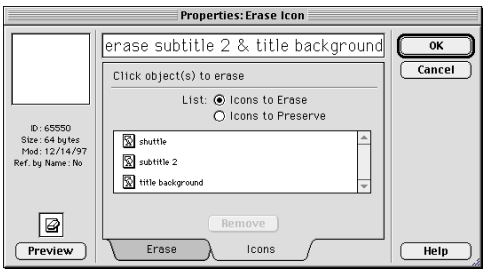
☐ **Press "Command/Control-J" to jump back to the flow line.**

☐ **Double-click on the final Erase icon titled "erase title background and subtitle 2."**

The most previous screen should appear containing just the shuttle graphic at the top right.

☐ **Click once on the shuttle.**

It should disappear. Also notice that this icon has been added to the top of the "erase list":



Click on the "Icons" tab to see this list.

☐ **Click "OK."**

This modifies the "programming" of this Erase icon to also erase the shuttle. You should now be back at the flow line.

And we should also change the name of this Erase icon to reflect the fact that an additional Display icon was erased.

Adding some sound

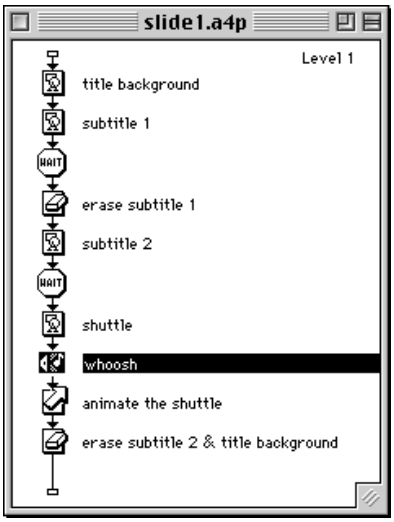
Sound icon

So far, our little slide show has come across as a silent movie. Let's give it some audio. The goal is to have the shuttle "whoosh" across the screen. This can easily be done with the Sound icon. The hard part is finding a good sound compatible with the way this icon works. Again, a nice "whoosh" sound is provided in the Rieber Clip Media resources that accompany this book.

☐ **At the flow line, drag a Sound icon to the flow line and deposit it between the "shuttle" Display icon and the "animate the shuttle" Motion icon.**

The location of this sound is crucial for the effect we are after. You'll soon see (hear) why this is the best location.

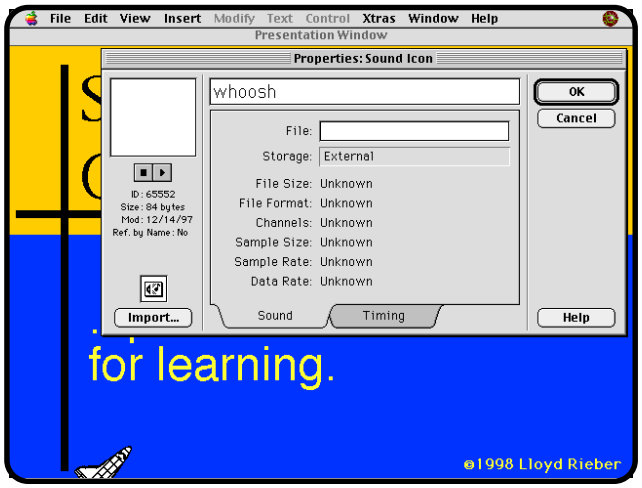
☐ **Title this Sound icon "whoosh."**



Once again, we'll just run the program and let Authorware automatically interrupt it when this undefined icon is encountered during the program's execution.

☐ **Run the file** (by pressing "Command/Control-R")**.**

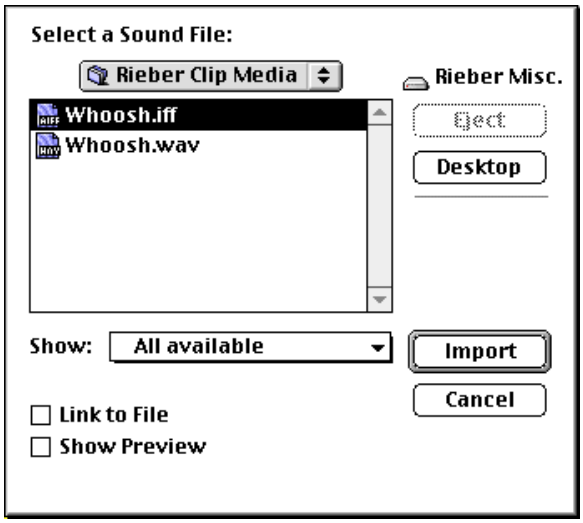
When this Sound icon is encountered, your screen should resemble the following:



This icon lets us play any pre-existing sound. The first step is to "Import" the sound.

☐ **Click "Import..."**

☐ **Point the dialog box to the "Rieber Clip Media" folder, click once on the audio file "Whoosh.iff", then click "Import."**

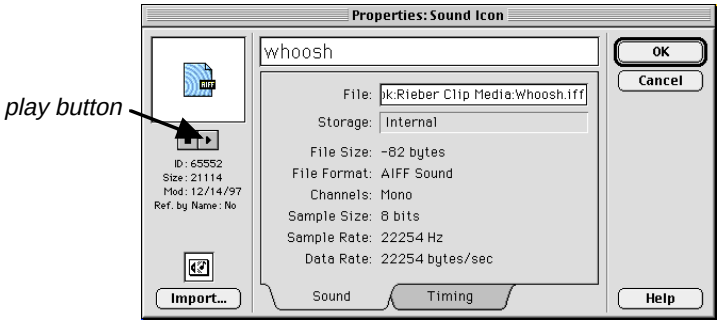


Authorware supports a variety of sound file types, with "Aiff and "Wave" being two of the most common.

These particular sound files came with Authorware — you probably have these on one of your resource disks.

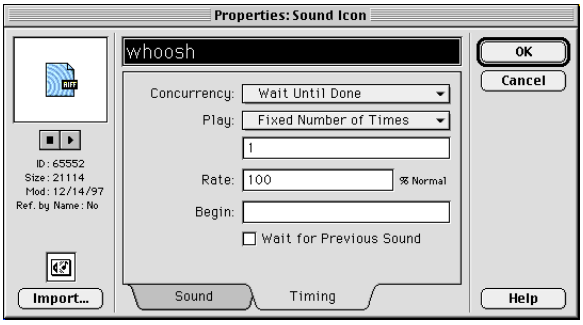
Let's test the sound.

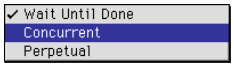
☐ **Click once on the play button (marked with a black triangle):**



Finally, we need to change from "Wait Until Done" to "Concurrent." That is, we want the next icon on the flow line (the Motion icon) to begin just as the sound starts. This option can be found in the "Timing" section of the dialog box.

☐ **Click once on the "Timing" tab.**





☐ **Click and hold on the words “Wait Until Done”, move the mouse down until the word “Concurrent” is highlighted, then release the mouse button.**

☐ **Click “OK.”**

Just one more test to make sure everything is working properly.

☐ **Restart the file (by pressing “Command/Control-R”).**

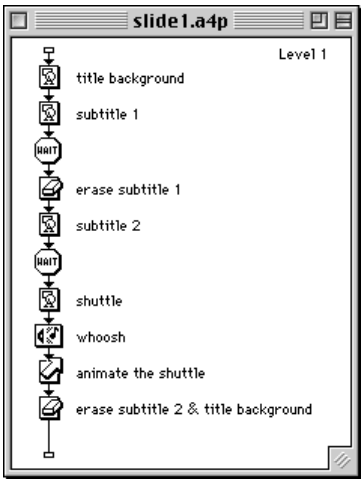
Be sure the entire title sequence runs smoothly. The animation and sound should seem well synchronized.

☐ **When done testing, press “Command/Control-J” to jump back to the icons.**

☐ **Press “Command/Control-S” to save the file.**

Grouping the icons in the title sequence

Your flow line should resemble the following by this point:



You may be wondering what happens when you run out of flow line. Yes, you could probably stretch the presentation window a little bit more, but ultimately you run out of room. There is no scrolling flow line in Authorware. You must use the screen space provided. As it turns out, Authorware lets you group icons in an infinite number of ways via the Map icon (sometimes known as the Grouping icon). You can group after the fact (such as the title sequence in our case), or you can drag Map icons to the flow line first in anticipation of grouping the program into meaningful chunks.

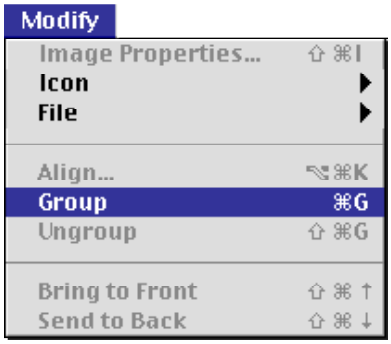
Let's group all of the icons currently on our flow line together within a Map icon called "title stuff." To group a series of icons already on the flow line, the first step is selecting the group. This can be done a couple of ways. Here's one way. You already know that clicking once on an icon selects it. However, if you hold down the shift key, you can multiple select two or more icons. Let's do it.

There are other ways to group. For example, you can draw a "select box" around a group of icons. You can also, when appropriate, choose "Select All" from the "Edit" menu.

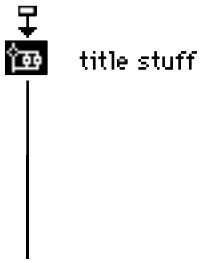
☐ **Press and hold down the "shift" key while clicking once on each of the icons on the flow line.**

When done, all of the icons should be highlighted:

☐ **Select "Group" under the "Modify" menu.**



A Map icon appears on the screen with the name “untitled.”

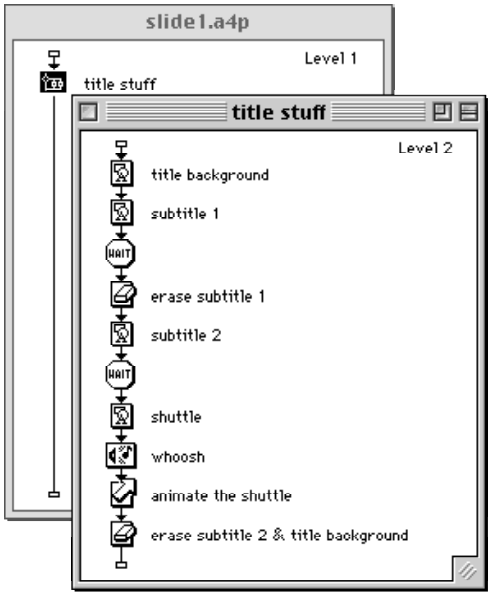


☐ **Change the name of the Map icon to “title stuff.”**

Now you have lots of room on the flow line. Where did everything go? Just double-click on the Map icon to find out.

☐ **Double-click on the Map icon “title stuff.”**

You will see that all of the icons are safe and sound inside. Notice, too, that there is a second presentation window with the label “Level 2” in the upper right hand corner:



There are no hard and fast rules for grouping. I suggest you use grouping as an aid to organizing your program. A modular program is easier to understand, troubleshoot, and revise.

There is also no limit to the number of groups within groups. Use grouping (and the Map icon) to effectively organize your program into manageable pieces.

Let’s finish this session by saving the file.

☐ **Press “Command/Control-S” to save the file.**

Time out

This is a good point to stop and reflect on what you accomplished. You've learned a great deal about Authorware in a short amount of time. It's a good idea to occasionally "take inventory" of your skills. You've learned how to program the computer by building a flow chart with the following icons: Display, Erase, Wait, Motion, Sound, and Map. You've learned how to construct an object-oriented display. You've experienced the pattern of jumping back and forth between designing on the flow line and running the program to test your design.

If you feel you only have partial understanding of the concepts and skills covered so far, you are advised to go through this section again from the very beginning. On the other hand, if you feel comfortable with what you have learned (even if you don't have perfect recall of all the steps), by all means proceed. You'll continue to hone these skills in the pages ahead.

This is also an excellent point at which to take a break. Here is the proper procedure for quitting (skip the next six steps if you are going to plow ahead):

- ☐ **Jump back to the flow line** (if you are not already there)**.**
- ☐ **Save your file one last time.**
- ☐ **Select "Quit" from the "File" menu.**
- ☐ **When back at the desktop, backup your file "slide1.a4p" by copying it to another disk.**
- ☐ **Go relax.**

To be honest, it is not necessary to be at the flow line when you save. If you save while running the file and then quit, Authorware will open the file exactly at that point. Novices often get confused when this happens, so I always recommend saving one more time at the flow line just before quitting.

Backing-up is the best protection against losing important work. The wisest developers always expect the disk or hard drive where their original files are stored to crash tomorrow.

What's next?

You have two options at this point. The most obvious is to finish the slide show project in this chapter. However, you are now well-prepared to begin the simulation and gaming projects starting with chapter 2. If you have been patiently waiting to get started building simulations and games and feel you understand what you have done up to this point, you can confidently proceed to Chapter 2. All of the remaining icons discussed in the rest of this chapter will be introduced in the chapters that follow (although in a different context).

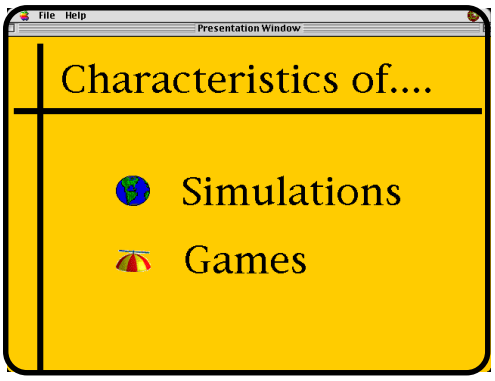
Well, almost all. The Framework icon will not be used in subsequent chapters.

When you are ready to resume this slide show project, you can just double-click on the file "slide1.a4p" (this assumes you are either on the same computer or one that has the same Authorware resources installed).

☐ **To resume this project, double-click on the file icon "slide1.a4p" to launch Authorware and open the file.**

Constructing the slide shows

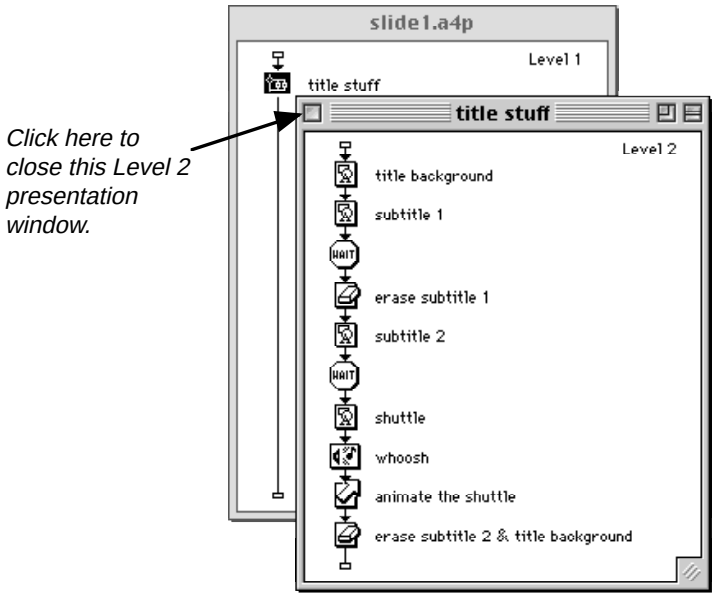
Time to finish this slide show. Since the presentation is about the role of simulations and games in education, you might be wondering which of the two topics is best presented first. One advantage of using a tool like Authorware is that we can program choices into the file. For example, let's make our slide show more flexible by giving the following "menu" right after the title stuff:



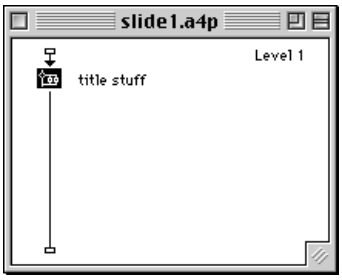
Then, we can click on either the word “Simulations” or “Games” (or the graphics in front) to go to a short slide show of each. After the slide show of that topic is over, the program will come back to this screen.

Let’s begin.

☐ **Close the Level 2 window “title stuff” by clicking once in the close box.**



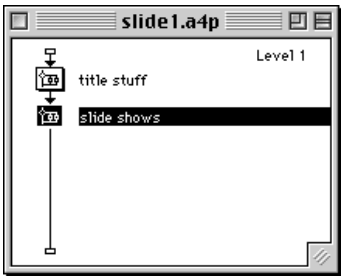
Your flow line should resemble the following:



Now that we know something about Map icons, let’s start by dragging one to the flow line into which we will put both our slide shows.

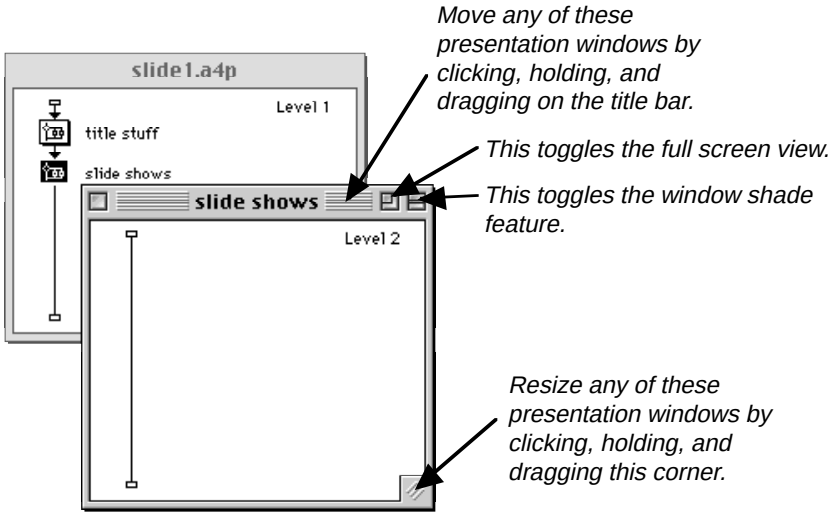
☐ **Drag a Map icon to the flow line and deposit it just below “title stuff.”**

☐ **Title this new Map icon “slide shows”:**



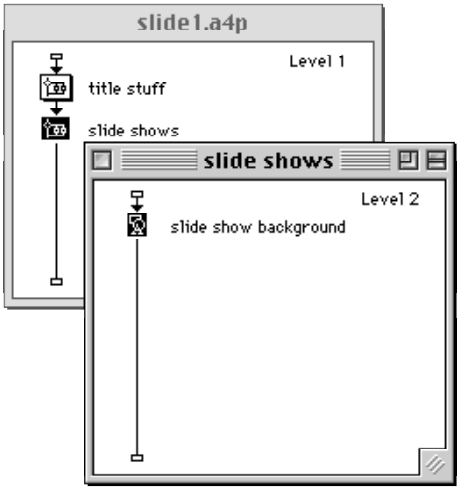
We will open this Map icon and then start building our slide shows inside of it.

☐ **Double-click on the Map icon “slide shows” to open it.**



Now let’s use one of the tricks we learned during the construction of the title — creating a separate display as a background for both of the slide shows.

☐ **Drag a Display icon to the Level 2 flow line and title it “slide show background.”**

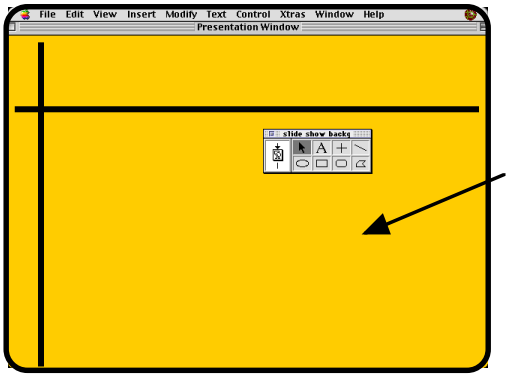


Let's go ahead and open this Display icon and construct a background for our slide show.

☐ **Double-click on the Display icon "slide show background."**

We will now start assuming you know how to use the Display icon's tool box to create object-oriented displays. Only abbreviated instructions will be given from here on indicating what objects to construct.

☐ **Construct the following display using the straight line tool and the rectangle tool:**



I constructed a rectangle filled in with a light yellow foreground to cover the entire screen.

☐ **When done, click in the close box in the tool box to go back to the flow line.**

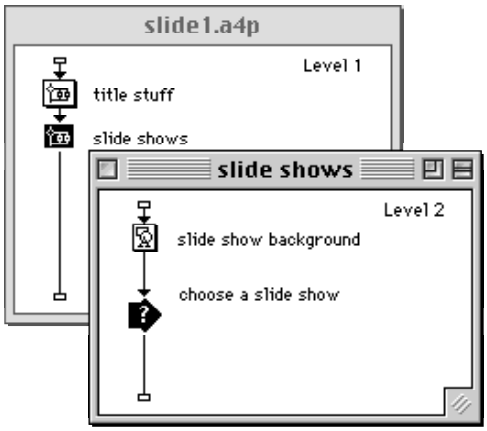
Now we want to program the menu choice of presenting either the slides about simulations or games. We do this with the Interaction icon. This icon provides a variety of means to let the user "participate" in your Authorware programs. This icon figures prominently throughout the rest of the book. We will avoid the temptation to explain what this icon does in depth here. Instead, the goal is to give you a quick experience in using the icon successfully. Again, understanding will come with experience.

The range of features and options in this icon is remarkable. Almost every kind of user interaction you can think of can be constructed with it. Buttons, pull-down menus, movable and clickable objects, character entry, etc. can all be done with this icon.

Interaction icon

☐ **Drag an Interaction icon to the Level 2 flow line and deposit it just below the Display icon.**

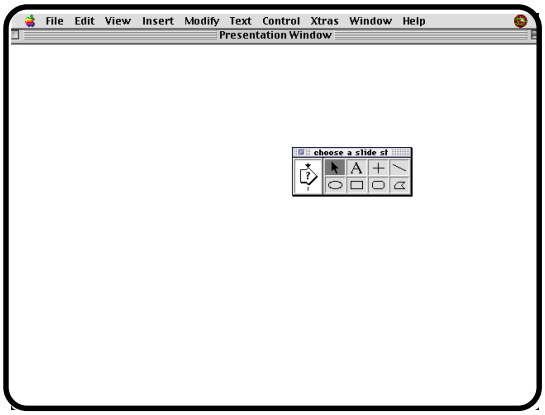
☐ **Title it “choose a slide show”:**



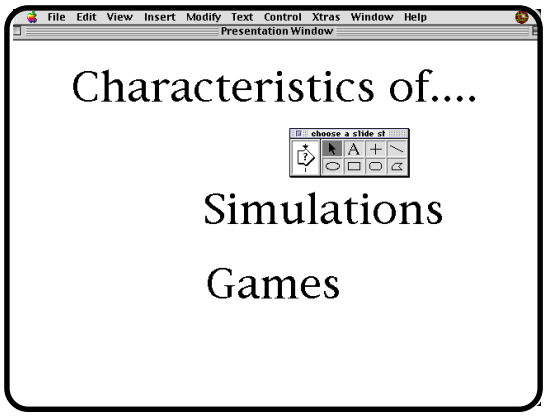
It’s useful at first to just think of an Interaction as a Display icon because it has all of the same display characteristics and functions. So, let’s double-click on it as though it were a Display icon.

☐ **Double-click on the Interaction icon.**

You get a display window that is identical to that found in a Display icon:



☐ **With the text tool, construct the following:**



Keep in mind that we want this text to line up appropriately with the slide show background we just constructed. You'll be able to fine tune the position of this text in just a moment.

Next, let's add a graphical symbol for both the concept of a game and simulation. Well chosen graphics can help to quickly convey the meaning of a concept. However, the more abstract the concept, the more difficult it is to find an appropriate graphic. For example, while you might easily decide that the graphic of an elm or oak tree well represents the concept of "tree," you might find yourself struggling with choosing a graphic for a concept like "justice." Probably graphics such as a judge's gavel or a balance scale come to mind because they are so common. Yet, how well do they really represent the concept? Concepts like simulations and games pose similar problems.

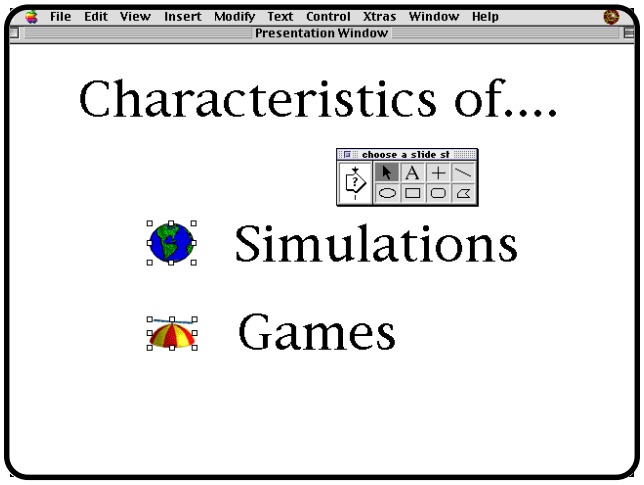
Yes, there are ways to have the previous background be displayed while constructing these text objects. These will be shown in other chapters. Unfortunately, the strategy of running the file and letting Authorware automatically open up undefined icons doesn't work with Interaction icons.

Let's use the graphic of the planet earth to represent a simulation and a "beanie boy" hat to represent a game (these can be found in the Rieber Clip Art resource). The planet earth can symbolize how simulations allow the user to visit and experience "other worlds" and the hat symbolizes the playful nature of games.

To be honest, I had difficulty finding appropriate graphics for these concepts.

If you feel you have more appropriate examples, don't be shy in using them instead.

☐ **Copy and paste each graphic, one by one, from the clip art file and into the Authorware program:**



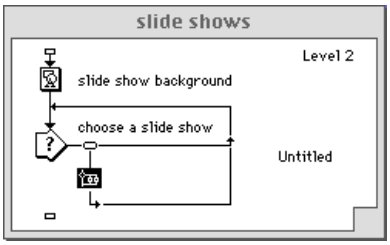
☐ **When done, click in the close box in the tool box to have Authorware accept this display and to return to the flow line.**

Adding hot spots to the display

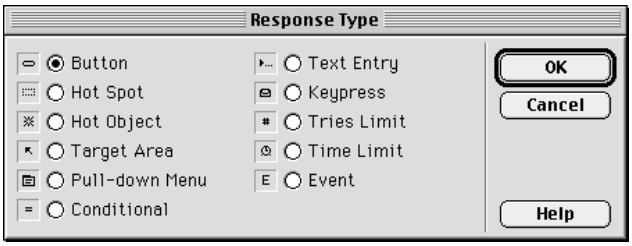
The next step is to add some interactivity. During our presentation, we want the opportunity at this point to either talk about simulations or games. Of the many response formats available with an Interaction icon, we are going to use two "hot spots," one for simulations and the other for games. A hot spot is simply an area of the screen that is sensitive to a mouse click. We can program any one spot to perform any task we wish.

☐ **Drag a Map icon to the flow line and deposit it just to the right of the Interaction icon.**

When you do, two things will happen. First, the flow line will change to the following:



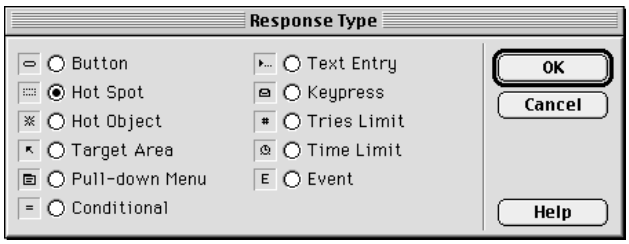
Second, the following "Response Type" dialog box pops open automatically:



This dialog box summarizes well the range of interactions possible with the Interaction icon. Some of these are obvious, such as "Button" and "Pull-down Menus," while others may be quite baffling, such as "Conditional." Again, we will not take the time now to explain each, but it pays to stop for a minute and consider all of the opportunities at this point.

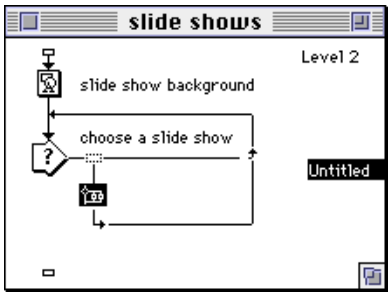
Notice that "Button" is the default choice. You are already very familiar with buttons through your everyday experiences working with the computer (e.g. the "OK" and "Cancel" buttons). Buttons are merely visible objects that are sensitive to mouse clicks. We want this interaction to be a hot spot. A hot spot is similar to a button with one exception — hot spots are invisible.

☐ **Click once in the radio button beside "Hot Spot."**



☐ **Click "OK."**

The flow line should now resemble the following:



You can think of this Map icon as a "branch" off of the Interaction icon. Let's change the title of this Map icon (or branch).

Though subtle, the graphic just above the Map icon has changed into a miniature hot spot (I like to refer it as a "baby hot spot"). This symbol not only signifies the response type for the branch, but will also lead to an important dialog box when double-clicked.

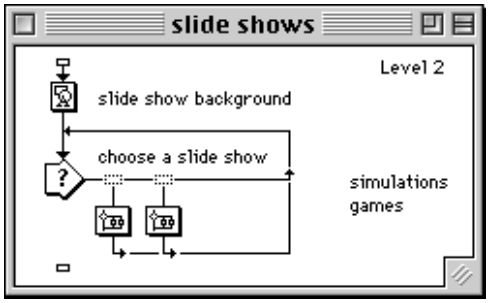
Each of the response types has a unique symbol (as shown in the response type dialog box).

☐ **Title this Map icon "simulations."**

Let's repeat this procedure to construct a second branch, also triggered by a "hot spot," for games.

☐ **Drag another Map icon to the flow line and deposit it just to the right of the Map icon "simulations."**

☐ **Title this second branch "games."**



You probably noticed that you weren't given the dialog box of response types this time. Authorware simply assumed that you wanted the same type.

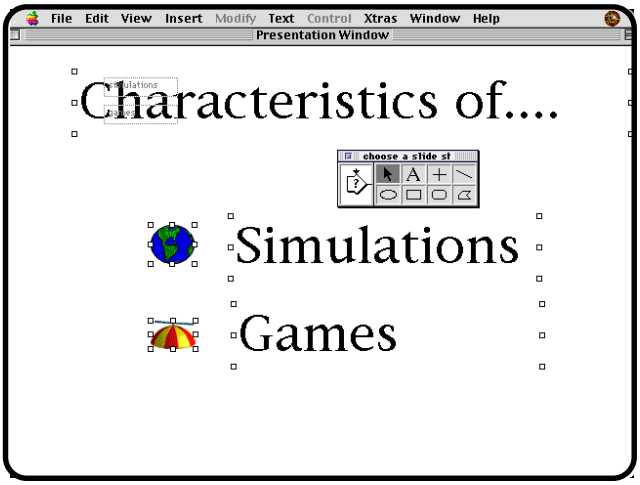
Later chapters will show how to change the response type after the fact. If you are desperate to know this now, double-click on the "baby hot spot" symbol to open a dialog box of options for this branch, of which changing the response type is one.

Positioning the hot spots

To position the hot spots correctly on the screen, we need to return to the display of the Interaction icon itself.

☐ **Double-click on the Interaction icon "choose a slide show."**

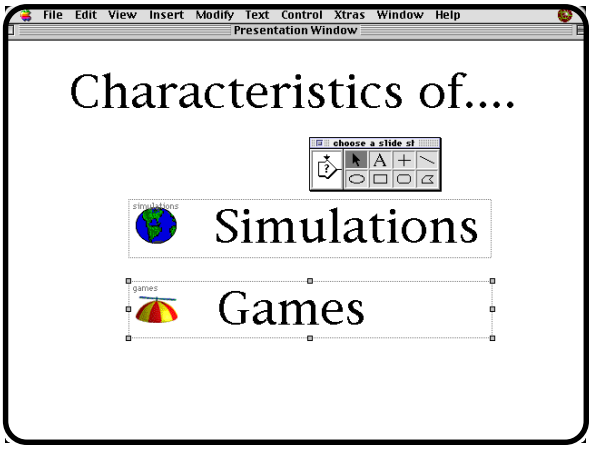
Notice the two small fuzzy boxes with the words "simulations" and "games" toward the top left of the screen:



These are the two hot spots we constructed on the flow line, one for each branch. They were automatically generated by Authorware. These can be resized by dragging on a handle and can be moved by dragging on any edge.

Warning! You need to carefully aim the mouse for either an edge or a corner of a hot spot. Otherwise, you may accidentally grab another object instead. (Remember, you can always select "Undo" from the "Edit" menu if this happens.)

☐ **Drag and resize the two hot spot over the appropriate words and symbols, as per the following:**



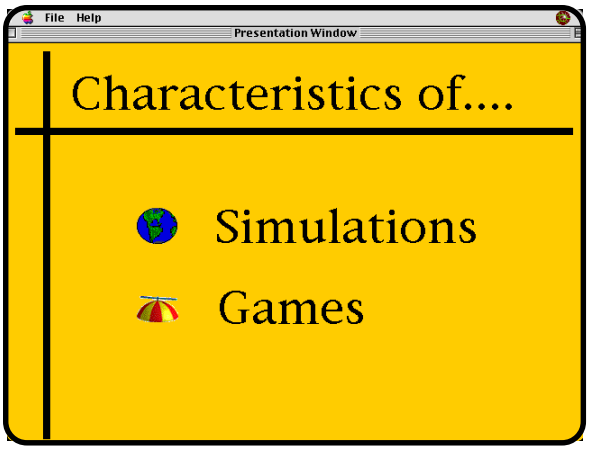
☐ **Click in the close box in the tool box when done.**

This sends you back to the flow line.

Let's run the program to check the alignment of the various displays.

☐ **Restart the file** (by pressing "Command/Control-R")**.**

☐ **Fine tune the position of the displays so that your screen resembles the following:**



Notice that the hot spots do not appear when the program is run. However, Authorware is now sensitive to mouse clicks in the areas defined by either hot spot.

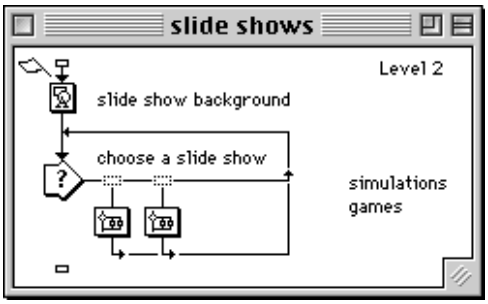
☐ **Press "Command/Control-J" to go back to the flow line.**

Using the start flag

When you run a file, you start at the very beginning of a program. For example, when you ran the program just a moment ago you were forced to go through all of the icons associated with the title even though you really just wanted to test the alignment of the icons "slide show background" and "choose a slide show." To save time (and preserve patience), Authorware gives you the opportunity to run the file from any specific location within the program via the "start flag." Let's see how this works.

Start flag

☐ **Drag the Start Flag to the flow line and deposit it just above the Display icon "slide show background":**



☐ **Select "Restart from Flag" from the "Control" menu** (or press "Option-Command/Control-R")**.**

Yes, this keyboard equivalent requires a bit more dexterity than the previous. Just press and hold down the Option key, then press Command/Control-R.

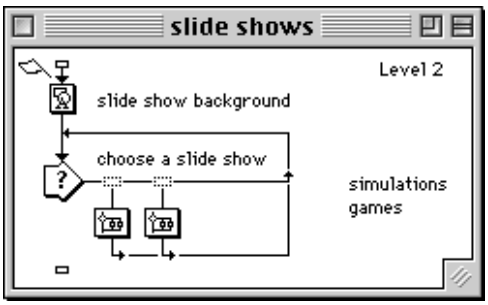
Authorware will begin executing the program starting with the icon it finds just below the start flag, thereby skipping all of the icons that precede it. Use this feature to your advantage.

☐ **Press "Command/Control-J" to jump back to the flow line.**

Adding slides

Time to finally add some slides to our slide show. Let's start by constructing just two slides for the part of our presentation extolling some of the virtues of simulations in education.

If you were brave enough to click on one of the hot spots when you were running your program a few minutes ago, you would have discovered that nothing much happens except for a momentary "flicker" of the screen. This is a good time to take a closer look at the flow line:

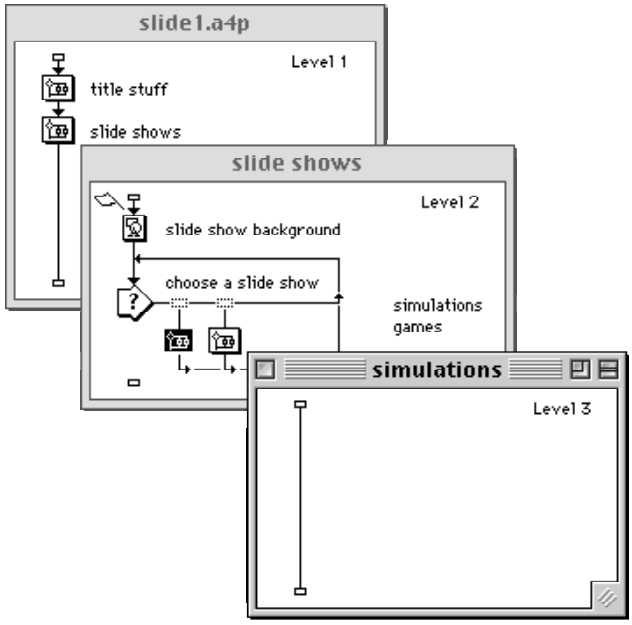


As will be often repeated in this book, the flow line is not just a pretty picture but an accurate representation of the way in which your program works. If the user clicks on the hot spot associated with the Map icon "simulations" the flow of the program is "sucked into" this Map icon. Whatever is inside this Map icon is then executed, whether that be one

icon or 100 icons. When finished, the flow exits the Map icon, hangs a right, then goes up and around back to the Interaction icon "choose a slide show." The same thing happens if the user clicks on the hot spot associated with the Map icon "games."

Of course, there is nothing inside these two icons as yet. That is about to change.

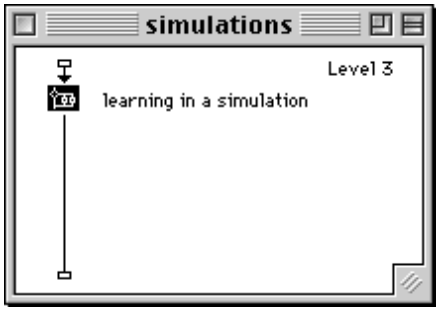
☐ **Double-click on the Map icon "simulations" to open it.**



A Level 3 presentation window opens (we opened a Map icon already nested in a Map icon). We are going to construct just two slides (you may want to construct more). The first will address the issue of learning in a simulation.

By now, you should be recognizing the value of using Map icons to organize your program. Let's drag one to this Level 3 flow line.

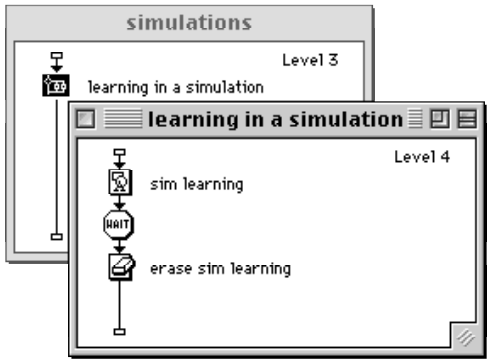
☐ **Drag a Map icon to the Level 3 flow line and title it "learning in a simulation":**



Now let's open this Map icon and add a sequence consisting of a Display icon, Wait icon, and Erase icon, similar to how we constructed the title group.

☐ **Double-click to open the Map icon "learning in a simulation."**

☐ **Drag a Display icon, Wait icon, and Erase icon to the Level 4 flow line and title them as per the following:**



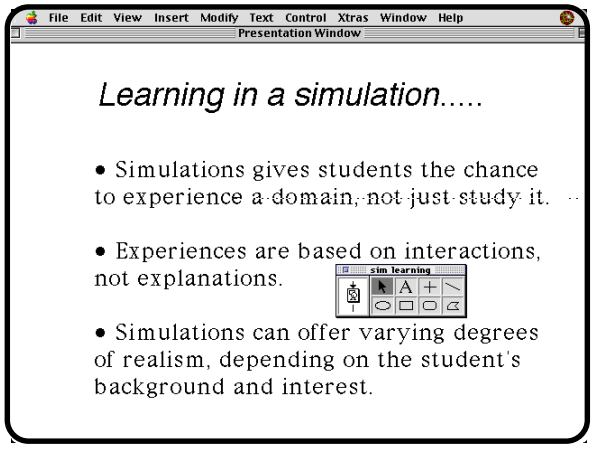
This group will display information, pause for the audience to read it (and to give us time to explain it), then erase it.

Let's go ahead and construct the display for "sim learning."

☐ **Double-click on the Display icon "sim learning."**

From here on, we are only going to insert text in order to keep the displays simple. You might want to come back to these slides later to add appropriate graphics.

☐ **With the text tool, construct the following display:**

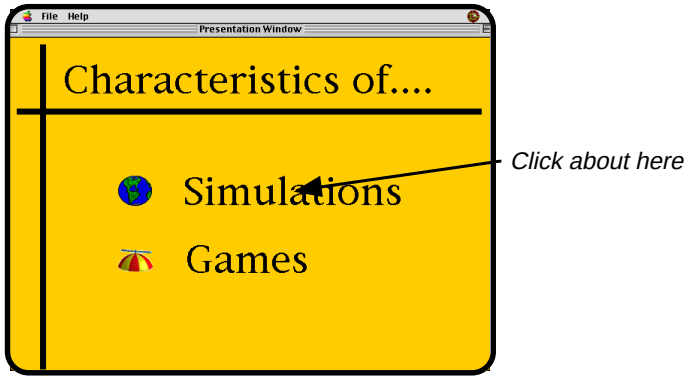


☐ **When done, click in the close box in the tool box to jump back to the flow line.**

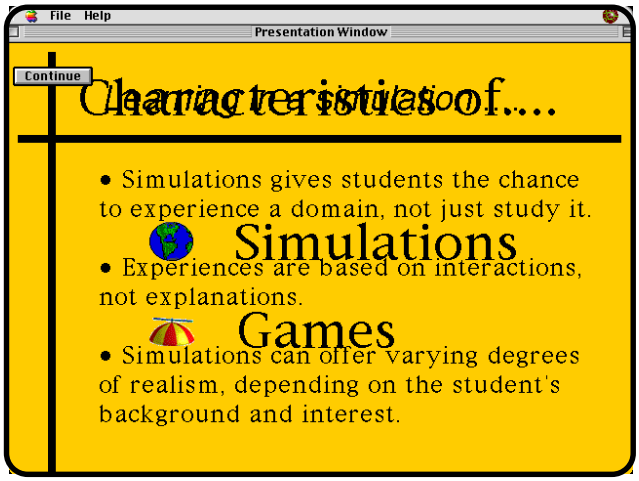
Of course, we need to have this text align properly with the "slide show background" display. Let's run the program from the flag to test it.

☐ **Select "Restart from Flag" from the "Control" menu** (or press "Option-Command/Control-R")**.**

☐ **Click on the screen area beneath the hot spot for "simulations":**



More than likely, your screen now resembles the following:

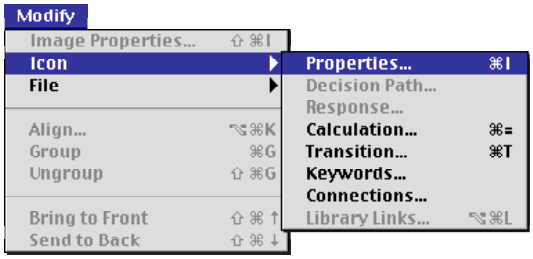


Yikes! What went wrong? Well, before you panic, take a good look at the screen. You'll notice that the only problem is that the menu of choices remained on the screen. To fix the problem, we simply need to erase the display associated with the Interaction icon "choose a slide show." There are a couple of ways to do this. The most obvious is to insert an Erase icon someplace. However, we will use this problem to introduce some of the automatic erasing features of Authorware.

First, let's go back to the flow line.

☐ **Press "Command/Control-J" to go to the flow line.**

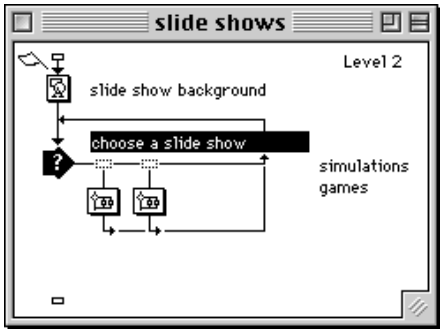
There are a variety of options associated with every Interaction icon. To get to this dialog box, you have two choices. First, you could click on the Icon (to select it), and then select "Icon" from the "Modify" menu, followed by selecting "Properties...":



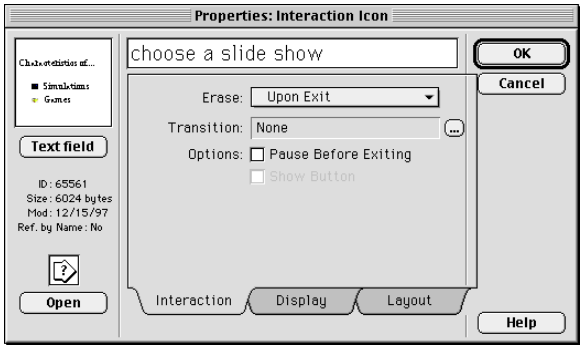
Another option would be to click on the icon (to highlight it) and then press the keyboard equivalent of Command/Control-I.

The second way to get to this dialog box is by Command/Control-double-clicking on the icon itself. This short-cut is the method preferred here.

☐ **Hold down the Command/Control key, then double-click on the Interaction icon "choose a slide show"** (on the Level 2 flow line)**:**

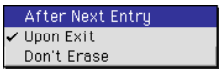


The following dialog box pops open:



As you can see, there are three parts to this dialog box as indicated by the tabs "Interaction," "Display," and "Layout."

Be sure that the "Interaction" options of this dialog box appear. If they don't, click on the Interaction tab. We are interested in the option beside "Erase:".



☐ **Click and hold on the words "Upon Exit," move the mouse up until the words "After Next Entry" are highlighted, then release the mouse button.**

The dialog box should now reflect the change:

By making this change, Authorware will now erase the display associated with the Interaction icon as soon as the user makes an "entry" — in our case, this means as soon as one of the two hot spots is clicked.

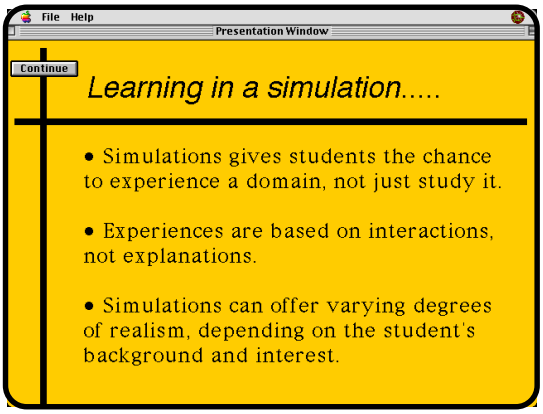
☐ **Click "OK."**

Let's restart the program from the flag to test this change.

☐ **Press "Option-Command/Control-R" to restart the program from the flag.**

☐ **At the menu, click on the hot spot beneath "Simulation."**

Your screen should now resemble the following:

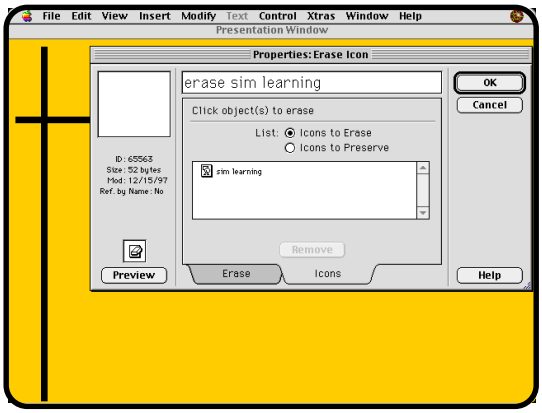


The difference being, of course, that the menu (but not the background) has been erased. Let's continue.

☐ **Click once on the "Continue" button.**

Since we never defined the Erase icon which follows on the flow line, Authorware automatically pauses the execution of the program and opens this Erase icon for us.

☐ **Click once on any of the text to erase the contents of the Display icon "sim learning."**



I clicked on the "Icons" tab to see the list of icons to be erased.

Be careful not to click on any object associated with the background.

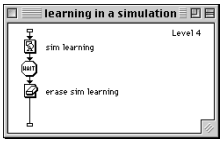
If you do, just highlight this icon's title in the erase list, then click "Remove."

☐ **Click "OK."**

You should now be back at the menu.

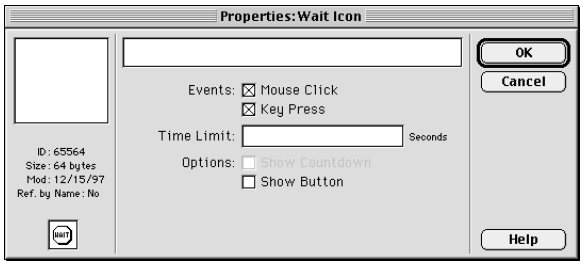
Now let's modify the Wait icon to accept any key press or mouse click and also not to display the "Continue" button.

☐ **Press "Command/Control-J" to jump back to the flow line.**



☐ **Double-click on the Wait icon inside the Map icon "learning in a simulation" (in the Level 4 flow line).**

☐ **Turn off the "Show Button" feature and turn on both "Wait for Mouse Click" and "Wait for Keypress":**



☐ **Click "OK."**

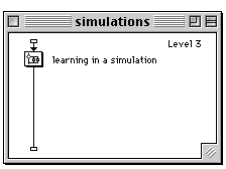
☐ **Restart your program from the flag to test it** (by pressing "Option-Command/Control-R")**.**

☐ **Jump back to the flow line** (by pressing "Command/Control-J")**.**

☐ **Save your file** (by pressing "Command/Control-S")**.**

Adding a second slide

Creating the first slide and making sure it works properly does take time. But now you can reap the benefits of your hard work because you can use the first slide as a template for your other slides by copying and pasting the Map icon containing this slide, then modifying the Display icon inside. We will create just one more slide as an example, but you can use this procedure to quickly create as many slides as you want.

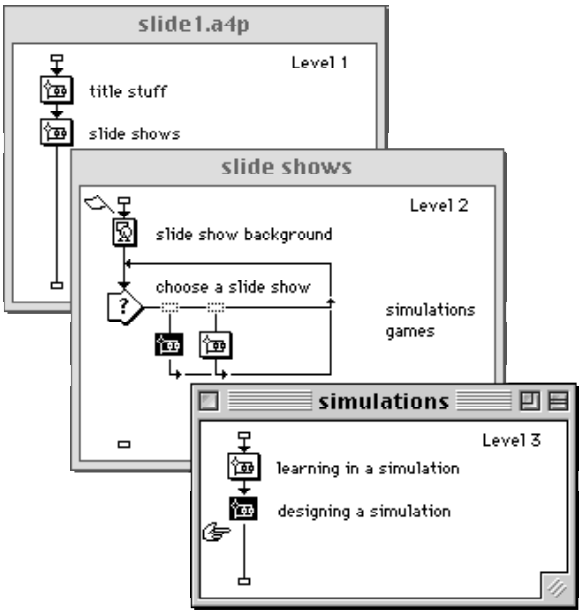


☐ **Copy the Map icon "learning in a simulation."**

☐ **Paste this Map icon just below and title it "designing a simulation."**

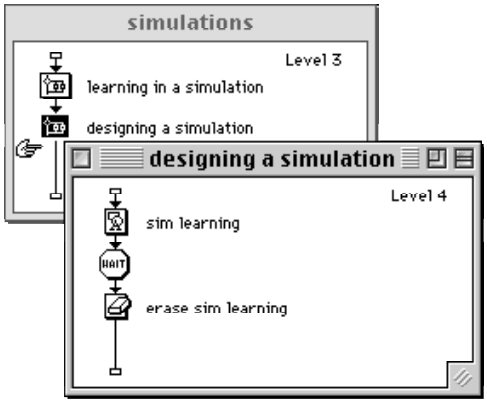
Hopefully, you are comfortable with the concept of copying and pasting icons by now. Here's a quick review of the steps: 1) click once on the Map icon "learning in a simulation" to highlight it; 2) select "Copy" from the "Edit" menu (or press "Command/Control-C"); 3) click once on the flow line just below the Map icon so that the paste hand appears; 4) select "Paste" from the "Edit" menu (or press "Command/Control-V"); and 5) be sure the title of this icon is highlighted, then type "designing a simulation."

Your screen should resemble the following:



Now let's change the contents the Display icon inside the Map icon "designing a simulation."

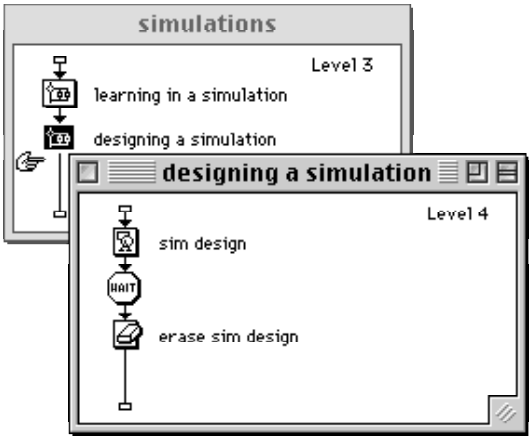
☐ **Double-click on the Map icon "designing a simulation" to open it.**



It's a good idea to rename the Display and Erase icons to reflect the new content.

But the program will work just fine if you don't. Think of all of the icon titles in this chapter as just comments to ourselves. Other chapters will demonstrate cases where icon titles are critical.

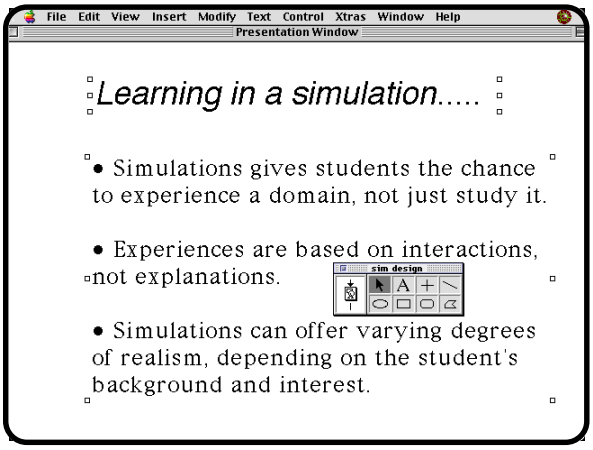
☐ **Rename the Display icon as "sim design" and the Erase icon as "erase sim design":**



Now all we need to do is change the contents of the Display icon "sim design."

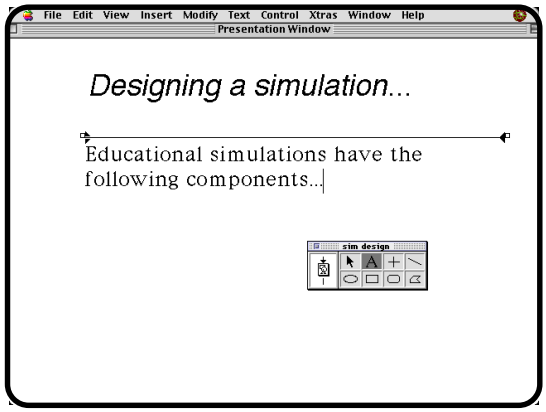
☐ **Double-click on the Display icon "sim design."**

The old content appears:

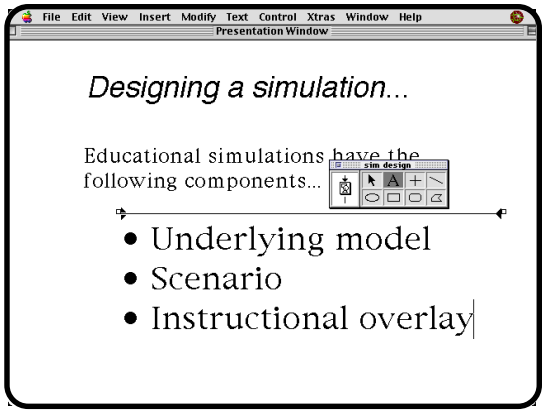


With the text tool, you can change the text while keeping its position, font, size, and format intact.

☐ **With the text tool, change the slide's title and body to the following:**



☐ **With the text tool, add the following text object:**



I used New York font with a size of 36 point.

☐ **When done, click in the close box in the tool box to go back to the flow line.**

Let's test our program by restarting it from the start flag.

☐ **Select "Restart from Flag" from the "Control" menu** (or press "Option-Command/Control-R")**.**

☐ **Run through the slide show dealing with simulations.**

Fine tune the position of any of the displays as you go.

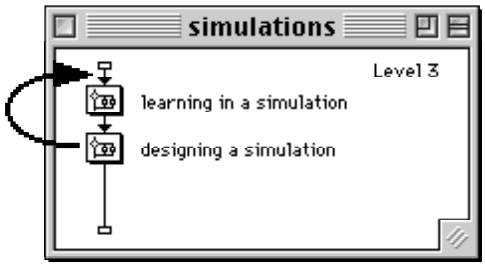
☐ **When done, jump back to the flow line** (by pressing "Command/Control-J")**.**

☐ **Save your file.**

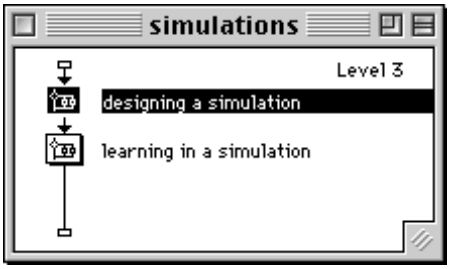
Rearranging your slides

Let's say you find out at the last moment that your audience will consist mainly of instructional designers. So, you decide that it would be better to start off the presentation with the design issues in the second slide. Fortunately, changing the order of the slides is very easy since we grouped each slide inside its own Map icon.

☐ **Drag the Map icon "designing a simulation" to the point on the Level 3 flow line just above the Map icon "learning in a simulation":**



All of the icons associated with this slide are now located first on the Level 3 flow line:



Let's test it.

☐ **Restart from the flag** (by pressing "Option-Command/Control-R")**.**

☐ **When done testing the program, jump back to the flow line** (by pressing "Command/Control-J")**.**

☐ **Save the file** (by pressing "Command/Control-S")**.**

A nonlinear slide tray

In creating the two slides for the simulation segment of the presentation, you may have noticed one limitation: it is not possible to return to a slide without first going through all the remaining slides and restarting the slide show. For a slide show of only two slides, this may not seem like a serious problem, but one with dozens of slides could spell disaster for the presenter. Frankly, up to version 3.0, the ability to go back and forth easily between a series of slides (or pages) was a bit of an embarrassment for Authorware. The one-way nature of the flow chart makes this simple need surprisingly complex. Although it's a problem easily solved, it requires a fair amount of true programming (with the help of a variable) to pull off a reasonable approach.

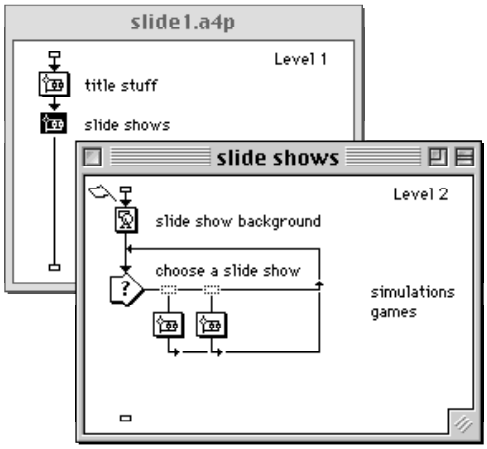
Fortunately, Authorware introduced two new icons with version 3.0 — the Navigate icon and Framework icon — which make short work of this and other nonlinear programming problems. In fact, our little problem of needing to go forward and backward during our slide show barely scratches the surface of the interactive potential of these two new icons. These two new icons will be of particular interest to multimedia designers wanting to explore the nonlinear learning environments referred to by either the terms "hypertext" or "hypermedia."

The introduction of the Navigation and Frameworks icons closes the gulf between Authorware and other hyper- authoring tools, such as HyperCard, Toolbook, and HyperStudio.

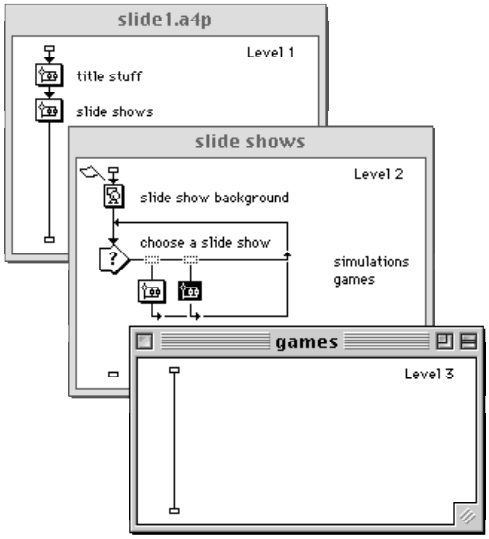
It's useful to remember that the prefix "hyper-" best translates as "link."

As always, let's begin at the flow line.

☐ **Close all but your Level 1 and Level 2 presentation windows:**



☐ **Double-click on the Map icon “games” to open its Level 3 flow line:**



Framework icon

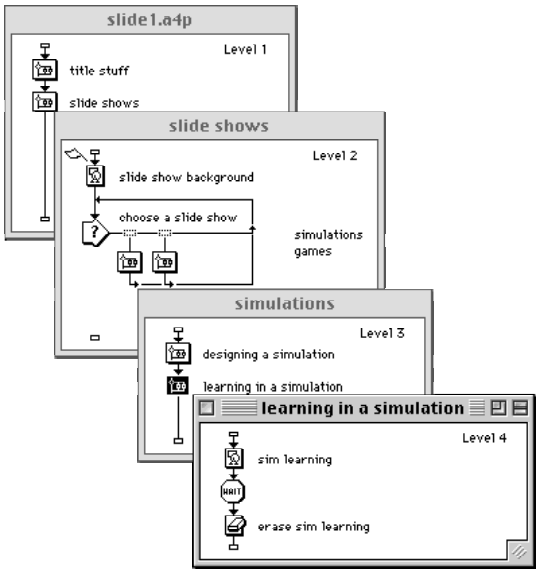
☐ **Drag a Framework icon to the Level 3 flow line.**

☐ **Title this Frame icon “gaming slides.”**

Icons are attached to a Framework icon in a similar way to Interaction icons. Like before, each of our slides will consist of a Display icon. However, the Framework icon will handle all of the work in managing the sequencing, pausing, and erasing of the displays. It also has some very interesting search capabilities built in as well.

We could start by attaching a new Display icon to the Framework icon and then build this display from scratch. However, let’s again use the first slide we made in the other section as a template to speed up our work. Since this is just a review of procedures already learned, they are presented here in abbreviated form.

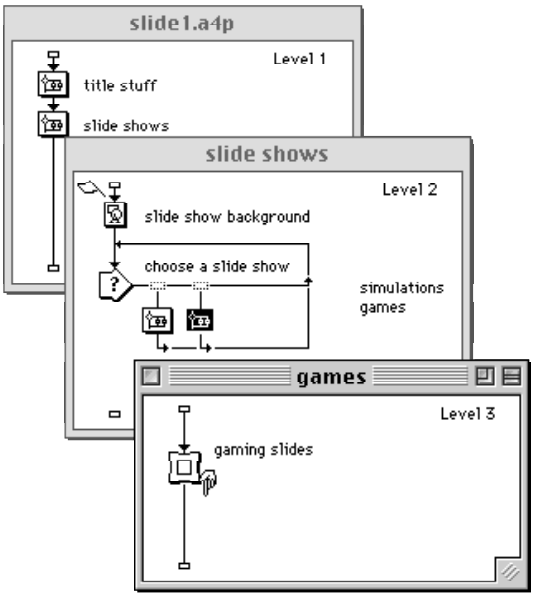
☐ **Open the Level 4 flow line associated with the Map icon “learning in a simulation”:**



☐ **Copy the Display icon “sim learning” to the clipboard.**

☐ **Go back and open the Level 3 flow line containing the Framework icon.**

☐ **Click once just to the right of this icon so that the paste hand appears:**



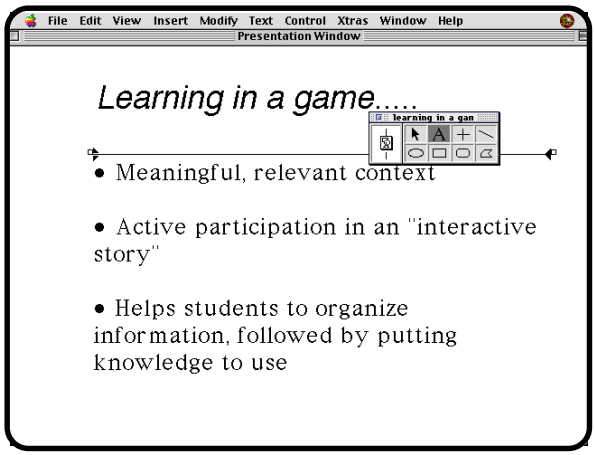
☐ **Paste the Display icon "learning in a simulation" at this point.**

☐ **Rename this Display icon "learning in a game":**

Now we can open this Display icon and change the text while keeping the text's location, font, size, and style intact.

☐ **Double-click on the Display icon "learning in a game."**

☐ **With the text tool, modify the text so that it matches the following:**



☐ **When done, click in the close box in the tool box.**

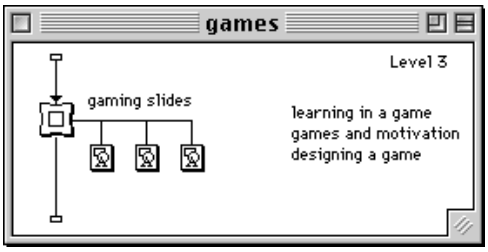
This sends you back to the flow line. Now we can use this Display icon as our template for the remaining two slides in this section. Again, we will copy this icon and then paste it twice.

☐ **Copy the Display icon "learning in a game."**

☐ **Click once to right of the Display icon "learning in a game" so that the paste hand appears.**

☐ **Paste this icon two times.**

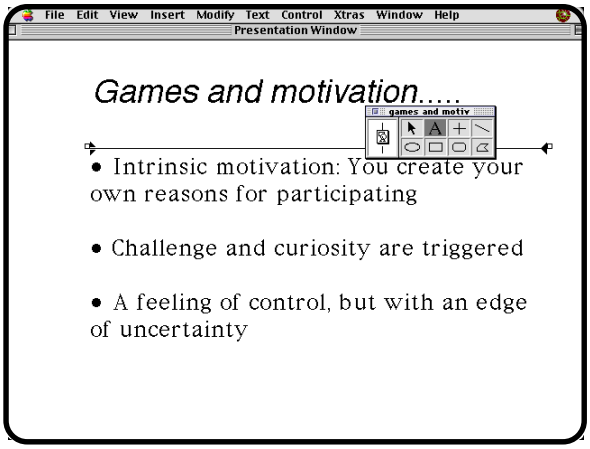
☐ **Rename these two Display icons “games and motivation” and “designing a game”:**



We will open and modify each of these two Display icons.

☐ **Double-click on the Display icon “games and motivation.”**

☐ **With the text tool, modify the text so that it matches the following:**

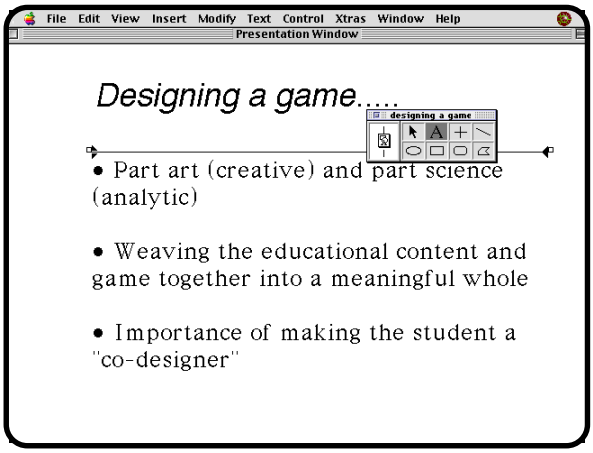


☐ **When done, click in the close box in the tool box.**

This sends you back to the flow line.

☐ **Double-click on the Display icon “designing a game.”**

☐ **With the text tool, modify the text so that it matches the following:**



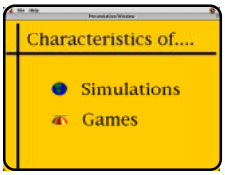
☐ **When done, click in the close box in the tool box.**

This sends you back to the flow line.

Testing the gaming slides

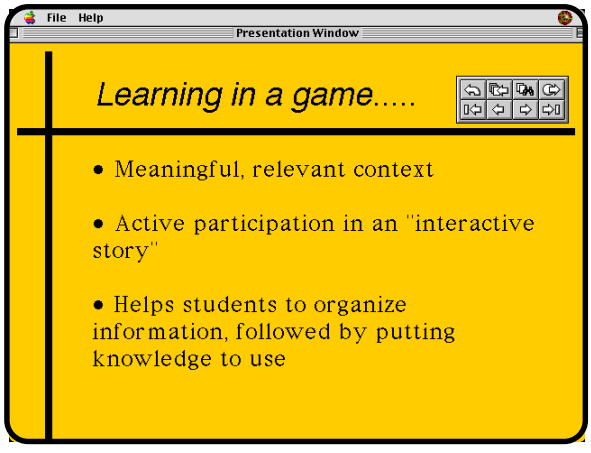
The best way to understand how the Framework icon works is by testing the program.

☐ **Select "Restart from Flag" under the "Control" menu** (or press "Option-Command/Control-R")**.**

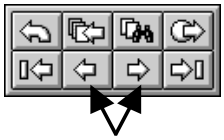


☐ **At the menu, click on the screen area under "Games."**

The first slide should appear along with a strange looking control panel (called an "entry pane") in the top right of the screen:



We will explore all of the functions of this entry pane in a just a moment, but for now let's just play with the forward and back buttons.



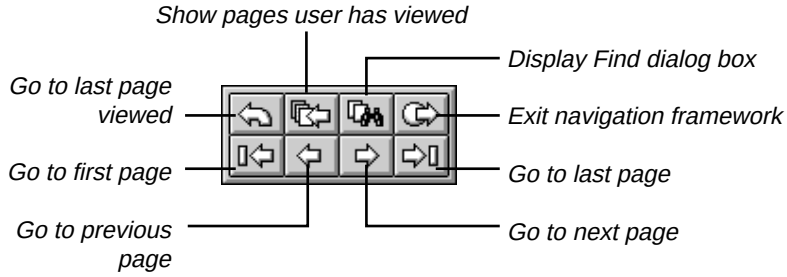
Click here and here to go back and forth through the three slides.

☐ **Click on the plain left and right arrow buttons in the entry pane to go back and forth through the three slides.**

You can think of your slides as a stack of cards you are holding in your hands. To go forward is like taking the card on top and putting it on the bottom. When you get to the last card, the first one is right behind. To go backward is like taking the card on the bottom and putting it on top.

Another often used analogy is a stack of cards in a rolodex.

Here is a brief explanation of the other buttons in the entry pane:



You are encouraged to play with these various buttons and to consult the Authorware manual to learn more about how they work (the "Display Find" button is particularly interesting). It is possible to move this entry pane to various positions on the screen, but you are advised not to do so at this point since the entry pane is really just a collection of various displays and buttons (it's easy to move any one piece, but rather time consuming to move them all).

The only other button we need to use at this point is the "Exit navigation framework" button.

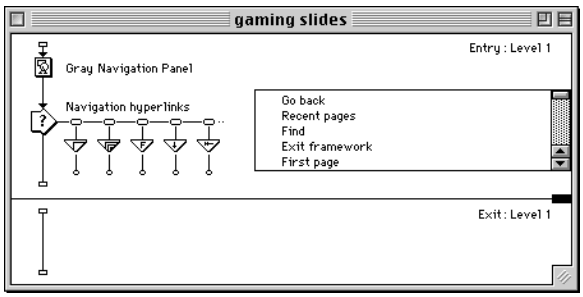
☐ **Click once in the "Exit navigation framework" button.**

This sends you back to the menu.

☐ **Press "Command/Control-J" to go back to the flow line.**

☐ **Save your file (by pressing "Command/Control-S").**

While the Framework icon leads to all kinds of interesting applications, we will not explore its use any further in this book. However, you may be wondering why we haven't used the Navigation icon even though it was highlighted at the start of this section. One reason is that you have been using it without realizing it. For example, if you double-click on the Framework icon, you will see the following:



As you can see, the Framework icon consists of a Display icon, Interaction icon, and Navigation icons. Consult Authorware's reference manuals for further information on how all this works (and how to modify it).

Improving the user interface

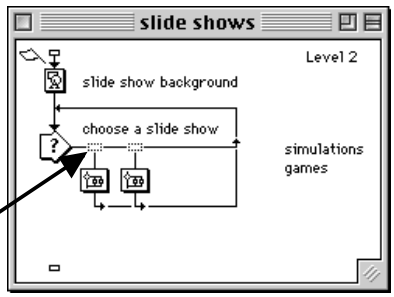
Let's consider ways to improve the way in which the program communicates its features and structure to a person not privy to details about how it was constructed. The term "user interface" is often used to define these sorts of design characteristics. For example, how would another person know that there are two hot spots on the main menu below the words "simulations" and "games"? Are there ways we could alert people to this? One way is to display printed directions directly on the screen. This is a reasonable strategy, but the additional text may also be distractive or appear as screen clutter.

Let's explore an alterative strategy common in many interactive multimedia materials: change the shape of the cursor from an arrow head into a symbol that conveys the idea that "here is a spot that is clickable." One such symbol is a picture of a hand with the index finger extended, ready to push a button:

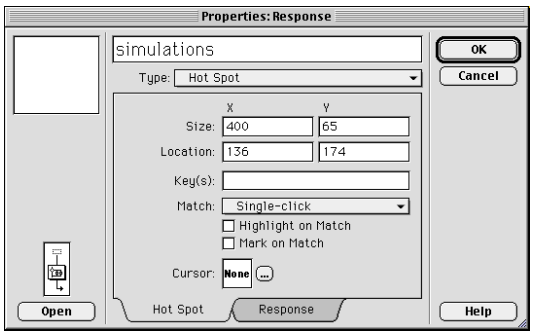
This symbol is often referred to as the "browser." Resist calling it "the finger" if you can!

Authorware makes it easy add this symbol to point out in a subtle way to users where hot spots are located.

☐ **At the flow line, double-click on the "baby hot spot" leading into the branch consisting of the Map icon "simulations."**



The following dialog box pops open:



Be sure you are viewing the "Hot Spot" options (otherwise, click on the Hot Spot tab).

The screen also changes to display the hot spot.

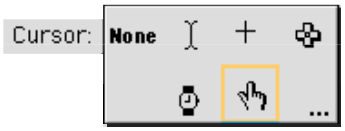
There are many options that can be explored within this dialog box. In fact, this dialog box will present different choices depending on the response type of the branch: options for a hot spot will be different than a pull-down menu or text response. Other chapters will explore these options further. For now, we are only interested in the option termed "Custom Cursor." As you can see, this is currently set to None. Let's change that.

☐ **Click and hold on the word "None" beside "Custom Cursor."**

This reveals the available custom cursor shapes.

Starting with Authorware 3.0, you can define and edit your own cursor shapes.

☐ **Select the hand symbol from the choices given:**



This symbol will then appear next to "Cursor:".

☐ **Click "OK."**

From now on, when the user moves the mouse pointer over the screen area defined by the hot spot, the cursor will change shape from the arrow head to the hand symbol.

☐ **Using these procedures, also change the cursor shape in the second branch consisting of the Map icon "games."**

☐ **Run the file to test this change.**

☐ **Jump back to the flow line** (by pressing "Command/Control-J")**.**

☐ **Save your file** (by pressing "Command/Control-S")**.**

Using digital movies

The slide show currently uses animation and sound at the end of the title stuff as an attention-gaining strategy. Let's consider another approach — the use of pre-existing digital movies. Authorware supports the use of both the QuickTime and PICS file protocols. Both are commonly referred to as "movies."

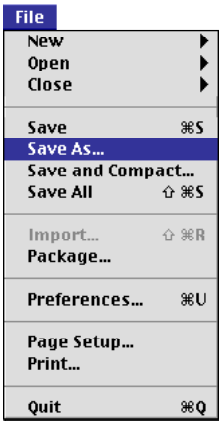
PICS animations offer one advantage over QuickTime movies — PICS animations can also be animated around the screen via the Motion icon. For example, a movie of a bicyclist pedaling a bike can be moved on the screen. In contrast, QuickTime movies only work in one place. The big advantage of QuickTime movies is the way in which they compress graphical information, thus allowing for a practical means of digitizing video.

In this section we are going to replace the animated sequence of the shuttle doing a "fly by" on the screen with a digital movie of the real space shuttle taking off from Cape Canaveral.

Of course, you may actually like the original animated sequence more. Therefore, we are going to save another copy of our slide show with the name "slide2.a4p". You will then have two versions of the slide show.

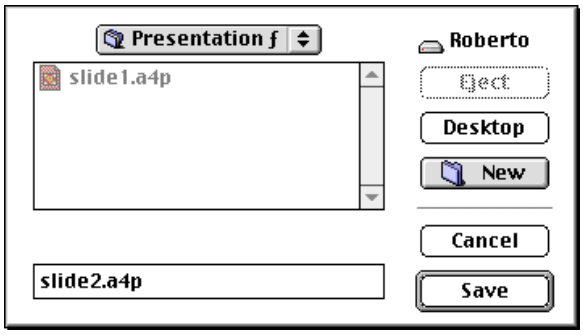
There are a couple of ways to proceed at this point. I usually prefer to quit Authorware, duplicate the file at the desktop, rename the file, then open and modify the renamed file.

However, another legitimate strategy that doesn't require quitting Authorware is to select "Save As..." again from the "File" menu in Authorware, rename the file, then save it.



☐ **At the flow line, select "Save As..." from the "File" menu.**

☐ **Change the name to "slide2.a4p".**



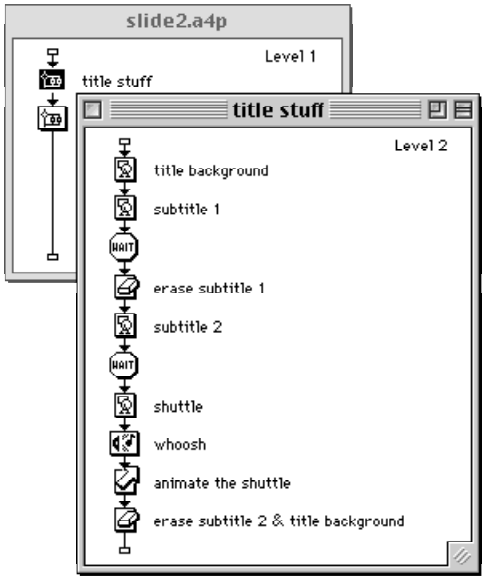
Be sure that the dialog box is also "pointing" to the appropriate folder in which the file should be saved.

☐ **Click "Save."**

From now on, all the changes you make will be to a second file called "slide2.a4p"; no changes will be made to the original file "slide1.a4p".

Let's now replace the animation and sound sequence with a digital movie. We need to open the Map icon called "title stuff."

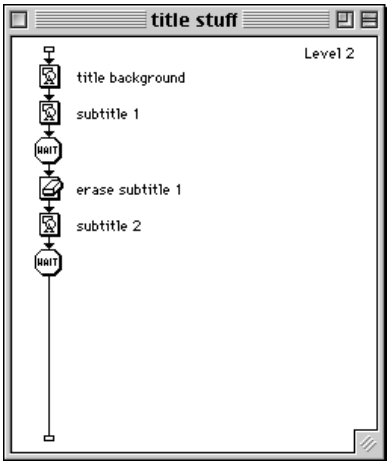
☐ **Double-click on the Map icon "title stuff" to open its Level 2 flow line:**



☐ **Delete the last four icons on the Level 2 flow line: "shuttle," "whoosh," "animate the shuttle," "erase subtitle 2 & title background."**

The simplest way to do this is by clicking on each once, then pressing the Delete key on the keyboard. When done, your Level 2 flow line should look like this:

A more efficient technique is to multiple select these icons (either by drawing a select box around them, or by holding down the shift key while you click on each one), and then delete them all at once.



You might be wondering why we are also deleting the Erase icon "erase subtitle 2 & title background." Well, let's consider the design strategy. As in the original file, we want the first and second subtitles to be displayed. However, after we click on the screen to continue, we now want *only* the second subtitle to erase (leaving the title background displayed). Next, the digital movie pops open on the screen and plays. Finally, the digital movie and the title background

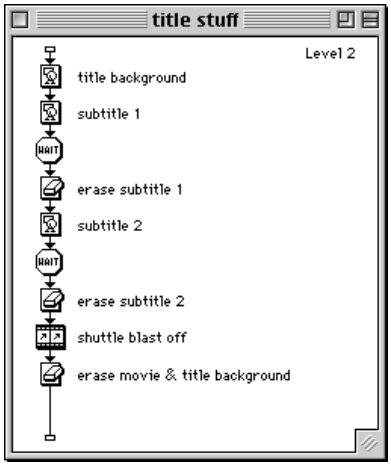
erases before the menu is displayed. While we could try to modify the existing Erase icon, I have found it much easier to delete old Erase icons and add new ones, then rely on the feature of Authorware automatically opening up undefined icons while running the program.

- ☐ **Drag an Erase icon to the bottom of the Level 2 flow line and title it "erase subtitle 2."**

- ☐ **Drag a Digital Movie icon to the bottom of the Level 2 flow line and title it "shuttle blast off."**

- ☐ **Drag another Erase icon to the bottom of the Level 2 flow line and title it "erase movie & title background."**

Your screen should look like the following:



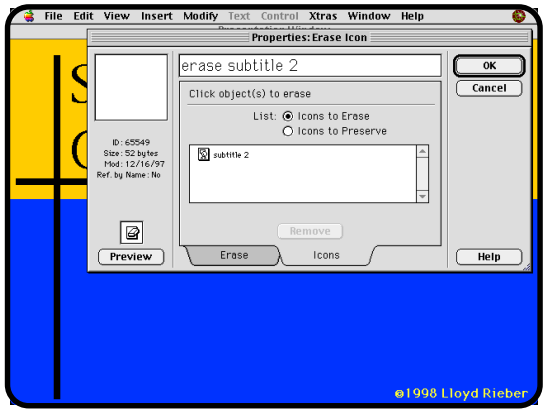
Important! We will be using a QuickTime movie called "shuttle launch." This movie is inside the Rieber Clip Media folder. If you do not have this movie, either stop at this point and get it, or find another suitable QuickTime movie to load in its place.

We will now run the file from the beginning, defining these new icons as we go.

- ☐ **Restart the file** (by pressing "Command/Control-R")**.**

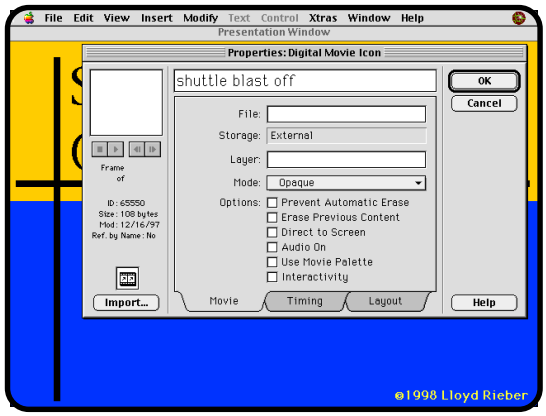
- ☐ **Step through the program.**

- ☐ **When the Erase icon "erase subtitle 2" automatically pops open, click once on the second subtitle.**



☐ **Click "OK."**

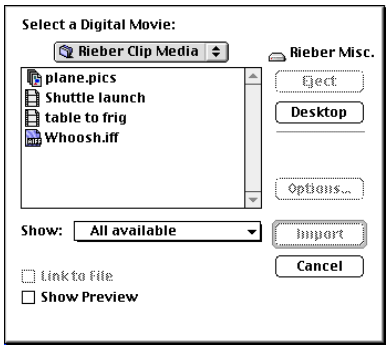
The next icon on the flow line is the Digital Movie icon. The purpose of this icon is to load (i.e. import) and play pre-existing digital movies. It likewise automatically opens.



☐ **Click once on the "Import..." button:**

Another dialog box opens for you to point to the movie you wish to open.

☐ **Point the dialog box to the Rieber Clip Media folder (or another folder containing the QuickTime movie you want to load):**

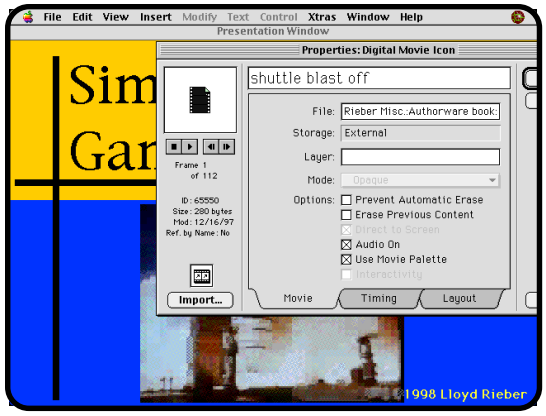


☐ **Click once on the title of the movie you want to use, then click “Import.”**

If you are using the Rieber Clip Media, choose “Shuttle launch.”

The first frame of the QuickTime movie appears on the screen. You can now move it anywhere you want.

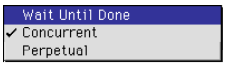
☐ **Move the QuickTime movie so that it is centered in the blue panel in the bottom half of the screen:**



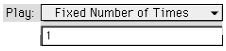
You will most certainly need to move the large dialog box out of the way often as you do so.

☐ **View the Timing options by clicking on the Timing tab.**

We need to make a few changes here.



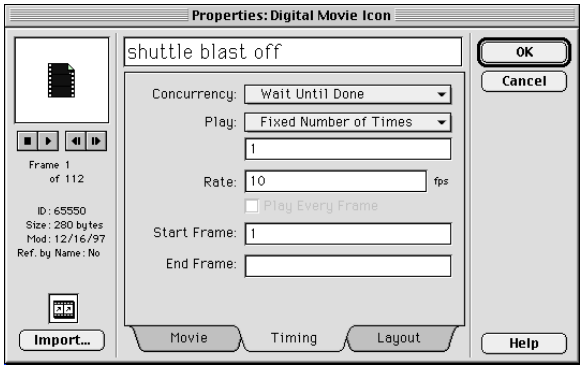
☐ **Change the “Timing” from “Concurrent” to “Wait Until Done.”**



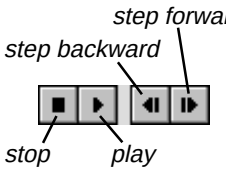
☐ **Enter “1” in the text box below “Play: Fixed Number of Times”.**

☐ **Enter a rate of “10” frames per second (fps).**

When done, the dialog box will look like this:



Changing from “Concurrent” to “Wait Until Done” assures that the program will not proceed to the next icon on the flow line until the movie has played to the end.



It's a good idea to test the movie at this point.

☐ **Click the play button in the movie controller.**

This movie consists of 112 frames. If the dialog box is set to play at 10 frames per second the entire sequence should last 11.2 seconds. (It should play only once, if it repeats, just click "Stop.")

Take a look at the other options in the Movie, Timing, Layout sections of this dialog box. Even though we won't be changing anything else, it's again a good idea to get familiar with the various choices here for future use.

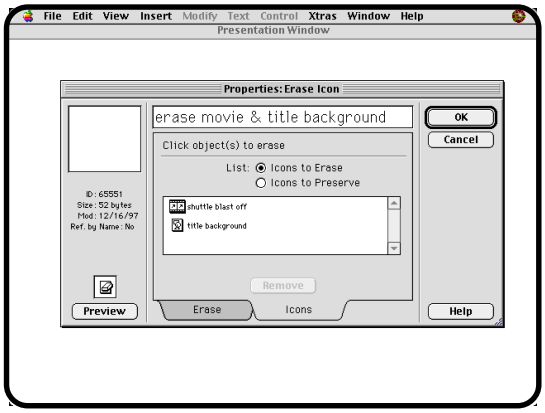
This movie includes audio. If you don't hear any sound, be sure that "Audio on" is checked (in the Movie options). If you still don't hear anything, be sure that the sound feature of your computer is turned on.

☐ **Click "OK."**

The movie should play in its entirety (with audio). When finished, the next icon on the flow line automatically pops open.

☐ **Click once on the "shuttle blast off" graphic to erase it.**

☐ **Click once on any screen object (such as the words "Simulations and Games") to erase the title background.**



☐ **Click "OK."**

The program should proceed to the screen menu. Let's run it again to test the smoothness of this sequence.

☐ **Restart the file** (by pressing "Command/Control-R")**.**

An astute presenter might say something like "It's time to launch this presentation" a fraction of a second before the digital movie appears.

Hopefully, the program advanced through the title sequence smoothly.

☐ **Jump back to the flow line** (by pressing "Command/Control-J")**.**

☐ **Save the file** (by pressing "Command/Control-S")**.**

You're done.

☐ **Package your file**

The last step in every Authorware project is a process called "packaging." This process creates another file that does not depend on the end user having the Authorware application. There is no need to rush into packaging. The packaged file is *not* intended to replace the original. In fact, it is imperative that you keep the original unpackaged file safe and sound (with lots of backups). You can always create as many packaged files as you want from unpackaged files, but there is no way to do the reverse.

I've created lots of teaching resources with Authorware. I'm usually packaging my final materials two or three minutes before class starts!

See Appendix A for general directions on how to package.

☐ **Knock 'em dead tomorrow with your presentation!**

Summary

You now know how Authorware works. You have produced a working slide show that uses almost all of the Authorware icons. You've learned how to construct and erase object-oriented displays in sophisticated ways to produce interesting effects. You've explored the world of multimedia by constructing a simple animation with an accompanying sound and compared it to the use of digital movies. You have created both linear and nonlinear sequences and have programmed choices into your slide show with the Interaction icon and Framework icon. You are now ready to begin building simulations and games!

Other projects

1. Add more slides to both the simulation and gaming sections.

2. Reduce the text density in the existing slides by finding and integrating appropriate graphics, animation, sound, and digital movies.

3. Revise the graphic design of the slide show. Don't be afraid to be creative! Of course, remember that "form follows function": your goal is to communicate, not impress.

Well, maybe impress them just a little.

4. Redesign the slide show to include examples of working simulations and games (such as the simulations and games you are about to construct in the following chapters). Consider where to embed buttons and hot spots that will lead to actual demonstrations of the simulation and gaming concepts presented here.

Chapter 2: Basketball Camp

Building a simple simulation with data-driven animation

This chapter will provide you with a solid experience in using variables and functions to create a fun and interesting project. Yes, it's true that these concepts rely on mathematics, but if you know how to add, subtract, multiply and divide, then you are more than ready to tackle the mathematics contained throughout this book. You might even find yourself enjoying mathematics a little more as a result because you will be using mathematics instead of studying it!

This chapter shows how to create from scratch a simple simulation that uses data-driven animation. This chapter provides a fun and easy introduction to variables and functions, mastery of which is crucial for building the remaining projects in this book.

In this session you will be saving the following file:

bball.a4p

This chapter assumes you are familiar with the structure of Authorware files and how to use and manipulate Authorware icons. Readers completely unfamiliar with Authorware are urged to complete Chapter 1 first.

The Problem

OK, to be honest this is not the world's best reason to create a computer simulation. Yes, it would make a lot more sense to have your team practice more in the gym. The point of this little story is to give you a context for doing the project. A good context provides meaning.

The basketball team you've been coaching has had some trouble on defense lately. Seems as though the players lack some fundamental knowledge about defensive strategies, the most important being this: "When guarding the offensive player, stay between the ball and basket!" You've decided that a little self-running computer simulation would help illustrate the principle. Maybe Authorware can help.

Introduction

So far, we've considered how to use Authorware as a presentation tool. We now turn our attention to how to create a simulation (or model) of a simple system. Our system is the game of basketball and has the following components: The context (the basketball court), one offensive player, and one defensive player. The underlying model of our system is based on one rule: wherever the offensive player goes, the defensive player goes to exactly the midpoint between the basket and the offensive player.

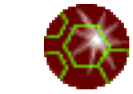
All educational simulations can be broken down into the following three parts: The scenario, the underlying model, and the instructional overlay,

In this example, the scenario is the basketball court and the underlying model is the defensive strategy. The underlying model can also be thought of as the "mathematical engine" on which the simulation operates.

The instructional overlay includes all information and features added to the simulation to guide learning, usually by making the key aspects of the underlying model explicit. In this example, there is no instructional overlay and would depend on someone (e.g. a teacher or coach) to describe what is going on.

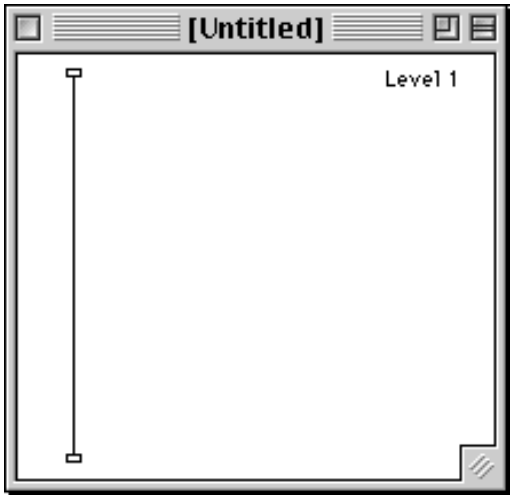
Getting started

The first step is to launch the Authorware application.



☐ **Start Authorware by double-clicking on the Authorware application.**

An empty level 1 flow line appears with the title "untitled":

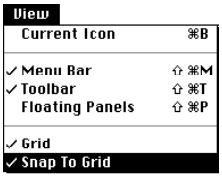


Setting up the displays

Next, let's set up our file to display the three main components of the simulation: basketball court, offensive player, and defensive player. Each of these will need to be contained in its own Display icon because of the animation that is planned for later — if we put all three objects in one Display icon, then it will not be possible to animate the two players separately.

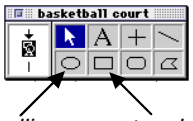
☐ **Drag three Display icons to the flow line and title them as per the following:**

☐ **Double-click on the Display icon titled "basketball court."**

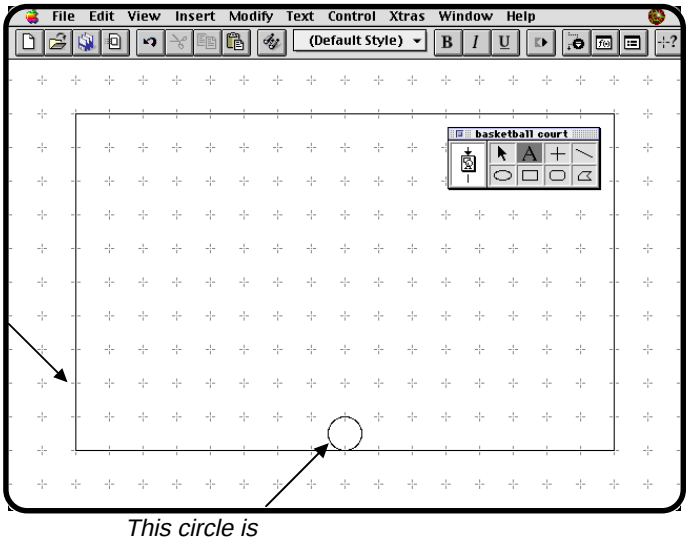


☐ **Turn on "Grid" and "Snap to Grid" under the "View" menu.**

Check marks will appear beside these options when they are on. These can be turned on and off at will. When the grid is turned on, objects move in 16 pixel increments. Visible grid lines are actually spaced 32 pixels apart. However, you can continue to use the arrow keys to move selected objects 1 pixel at a time even when the grid is on.



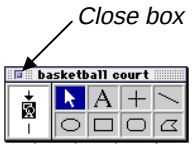
☐ **Draw the following basketball court with the Rectangle tool and Ellipse tool:**

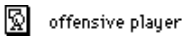


Obviously, we are not after commercial quality art work here. Remember, the key word through this book is "prototype." You can always go back and replace the crude graphics drawn here with more polished versions later.

Draw the court floor first with the rectangle tool, then draw the hoop with the ellipse tool. It's very important to draw the hoop exactly at the midpoint of the court floor because the animation we will construct later will only make sense to the user if the graphic is drawn properly.

When the grid is turned on, it's fairly easy to draw perfect circles. However, when the grid is turned off, you can still make perfect circles and squares by holding down the "shift" key as you use the rectangle or ellipse tool. This is a standard convention that works with most graphics applications.





☐ **When finished, click on the close box in the tool box to go back to the flow line.**

☐ **Double-click on the Display icon titled "offensive player."**

☐ **Copy and paste clip art of a suitable graphic, such as this.....**

This graphic can be found in the Rieber Clip Media collection.

(Consult chapter 1 on how to import clip art.)

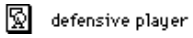
.....or just create a graphic "placeholder" using the rectangle and text tools, such as this:

I create graphic placeholders like this all the time. It allows me to quickly create the structure for a file without losing my train of thought. I hate getting bogged down looking for graphics. If you are working on a development team that includes a graphic artist, this is a great way to show the context in which the graphics will be used.

Put the offensive player graphic anywhere on the left hand side of the screen towards the bottom of the court.

I have seen too many people take too many hours looking for that one "perfect" graphic. It's kind of like taking a whole day just to write the title of a term paper! Resist the tendency to turn the hunt for graphics and other multimedia components into just another creative form of procrastination.

☐ **When finished, click on the close box in the tool box to go back to the flow line.**



☐ **Double-click on the Display icon titled "defensive player."**

☐ **Copy and paste clip art of a suitable graphic, such as this:**

This graphic can also be found in the Rieber Clip Media collection.

Again, however, feel free to simply put in another graphic place holder.

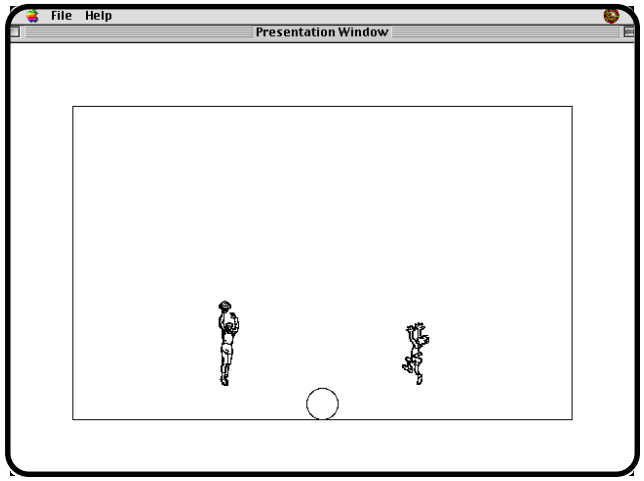
Put the defensive player somewhere on the right hand side of the screen towards the bottom of the court.

☐ **When finished, click on the close box in the tool box to go back to the flow line.**

Let's see if everything looks to be in their proper place by restarting the file.

☐ **Select "Restart" from the "Control" menu.**

The following graphics should appear on your screen:



What's most important is to establish the fantasy context of a basketball court. If the graphics don't adequately trigger the fantasy for the user, then you need to find or create some that do. By triggering the fantasy, you also trigger the user's imagination. (The most elaborate and realistic graphics don't necessarily do the best job of establishing the fantasy.)

Remember, when you "Restart" your program, Authorware executes your flow line, starting at the top. Even though it seemed as if all three graphics appeared simultaneously, in reality, the basketball court was displayed first, followed by the offensive player, followed finally by the defensive player.

It's no big deal if your players are in slightly different spots. However, to "fine tune" the position of the court or either player, just click and hold on the graphic and move it where you want it.

Remember also that you can double-click on any graphic while running your file to have Authorware open the corresponding Display icon. *The Display icon must be opened in order to edit the contents of the display.* (Hint: Look for the tool box; if you can't see it, then you know you are not in "edit mode."

☐ **Select "Quit" from the File menu (or press Command/Control-Q) to go back to the flow line.**

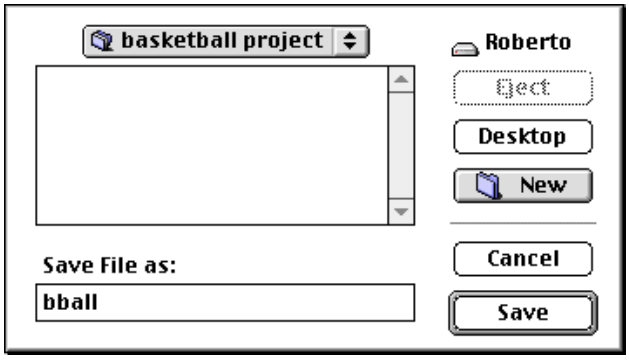
Platform Alert!

As mentioned in chapter 1, Macintosh users use the Command key and Windows users use the Control key for all keyboard shortcuts. (No further reminders will be given in this chapter.)

Some other useful keyboard commands are Command/Control-J to "jump" back to the flow line or Command/Control-B to show the current icon (this last one is very handy when your file gets complicated with lots of levels).

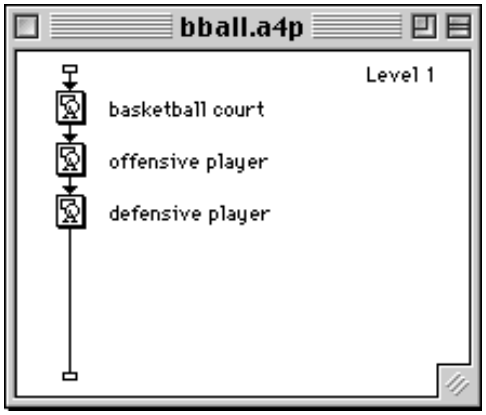
We haven't saved our file yet, so let's do that now.

☐ **Select "Save as..." from the File menu.**



☐ **Save your file with the name "bball" to the folder of your choice.**

You'll also notice that the extension ".a4p" has been added to your file name automatically:



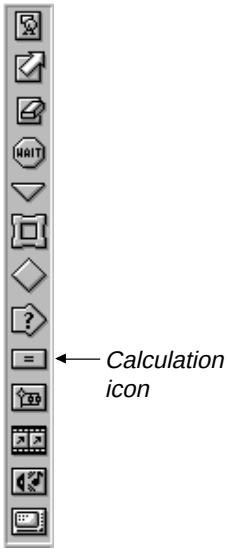
From here on, remember to save your file periodically using the "Save" command from the "File" menu. (Only use "Save as..." when you want to save another copy with a different name or to a different disk location.)

Setting up the data-driven animation for the offensive player

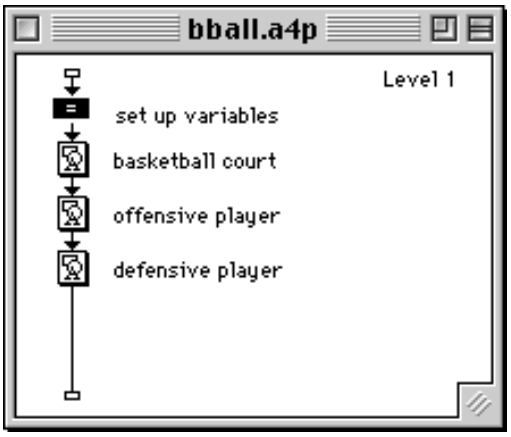
As you know, the goal is to have the offensive player move to random spots on the basketball court. The defensive player is then supposed to move to the exact midpoint between the offensive player and the hoop. In this section, we focus solely on the animation for the offensive player. In order to do this, we will need to set up two variables, one for the player's horizontal position and the other for the vertical position. (We will need to do the very same thing later for the defensive player.)

So, let's first create the necessary variables within a Calculation icon. Although the exact location of this icon does not matter so long as it comes before the animation icon, let's put it at the very beginning of the file.

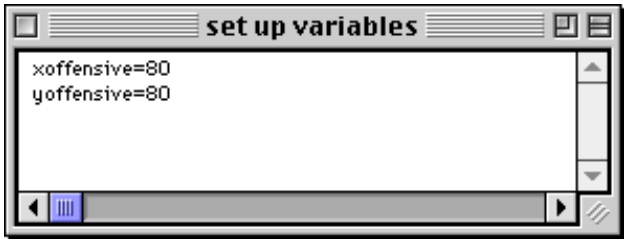
Creating variables



☐ **Drag a Calculation icon to the top of the flow line and title it "set up variables."**



☐ **Double-click on the Calculation icon "set up variables" and type in the following inside the icon's window:**

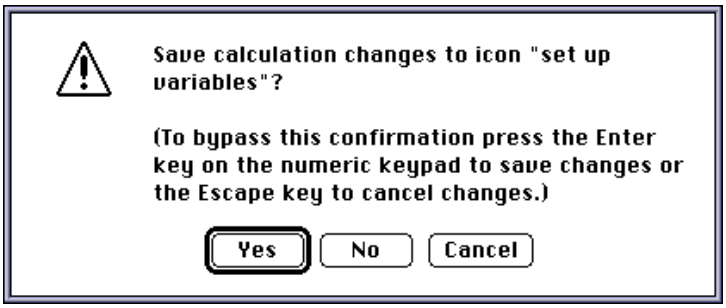


These two lines create two variables and assigns a value to each. The variable “xoffensive” will control the horizontal position of the offensive player and “yoffensive” will control the vertical position. (The letters x and y are typical mathematical notation for the horizontal and vertical dimensions.)

Even though mathematicians insist on using x and y for labeling 2-dimensional space, it sure seems as though h and v make more sense (short for horizontal and vertical). In fact, I usually use h and v in my own programming. Part of the trouble people have with formal mathematics is that the symbolism seems awkward, arbitrary, and difficult to understand.

☐ **When finished, click on the close box in the Calculation icon’s window.**

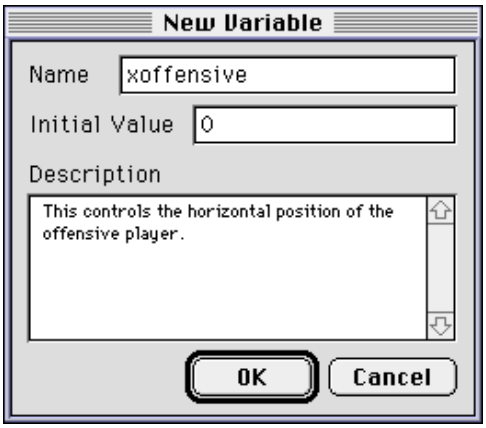
The following dialog box will appear:



☐ **Click on the “Yes” button.**

You should immediately see the new variable dialog box. You will get this dialog box *every* time Authorware encounters a new variable.

☐ **Type a zero beside “Initial Value:” and type a description similar to the one below:**

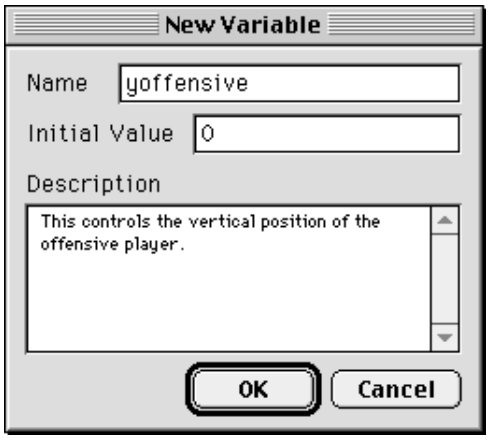


Be sure to type a “0” (zero) and not a capital “O”.

☐ **Click on “OK.”**

You should then immediately see another dialog box for the other variable you created.

☐ **Like before, type a zero beside "Initial Value:" and type a description similar to the one below:**

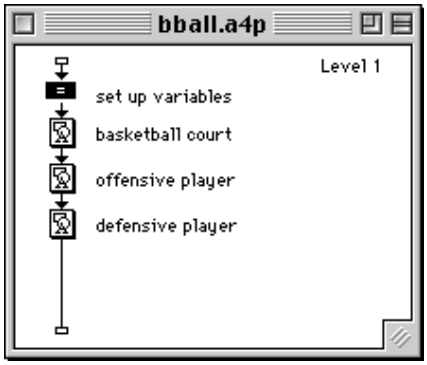


☐ **Click "OK".**

Actually, typing the zero and the description are not required, but they are good habits to get into. If no initial value is entered, Authorware will "figure out" the initial value based on how the variable is used.

There are three kinds of Authorware variables: **numeric**, **character**, and **logical**. Numeric variables store numbers, such as 54 and 1003. Character variables store strings of characters, such as the name "Lloyd." Logical variables simply store the value of true or false. Obviously, any variable upon which we want to perform some mathematical operation will need to be a numeric variable. (That is why it was so important to make sure you typed in a zero and not a capital O a few moments ago. You can't divide a letter by 2!) As already mentioned, Authorware "figures out" what kind of variable it is based on how it is used.

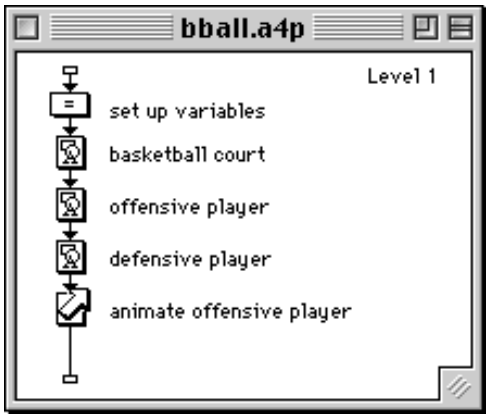
You should now be back at the flow line:



The important point to remember here is that when you close the Calculation icon's window, the commands you've typed inside it become part of the flow line and will be executed, in order, when the Calculation icon is encountered during the "flow" of the program.

Creating data-driven animation

☐ **Drag a Motion icon to the flow line and title it "animate offensive player."**



__Note__: This icon is called the "Animation icon" in earlier versions of Authorware. A much better name in my opinion. One reason for the change was to help users distinguish between the two different kinds of animation possible in Authorware: the motion of objects on the screen (via the Motion icon) versus the frame-by-frame animation available through QuickTime and PICS movies (via the Digital Movie icon).

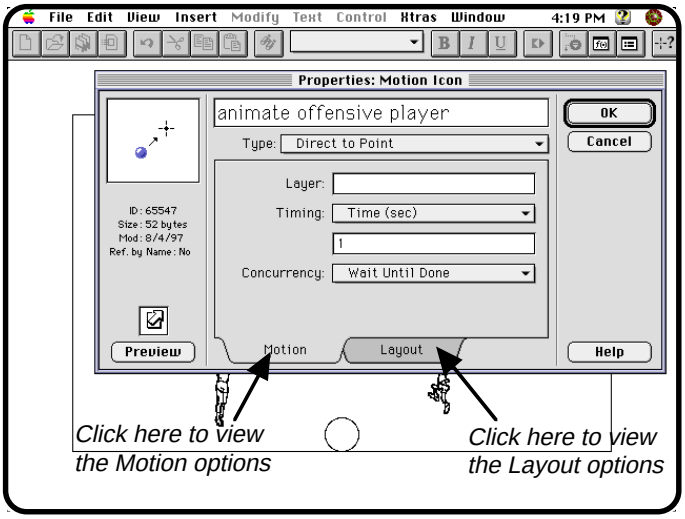
☐ **Select "Restart" from the "Control" menu.**

You might be surprised as to why we did not just double-click on the Motion icon. The answer has to do with the way in which the Motion icon works. When it is opened, the last thing displayed on the screen will magically reappear. In our case, we want to ensure that the offensive player and the basketball court both show up — even though we are only going to animate the offensive player at this point, the basketball court will be an important guide, as you will see.

By restarting the program, the first four icons are executed. When Authorware encounters any icon that has not been defined (such as our Motion icon), it will always pause the execution and automatically open the icon.

I take advantage of this feature in Authorware all the time. You will find that it is a tremendous time saver.

The following should appear on your screen:



Notice that the title we gave to the Motion icon appears at the top of the dialog box, making it very clear what this icon is supposed to do — you help yourself by naming your icons well.

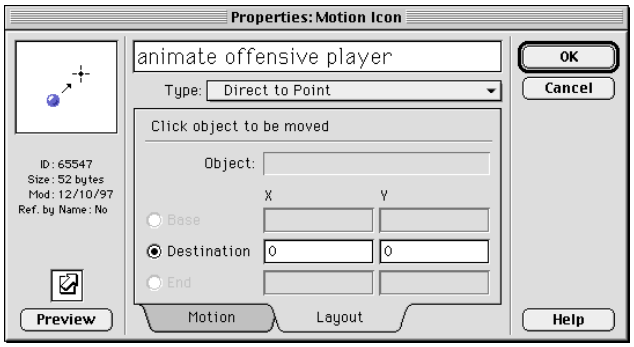
You can move the dialog box anywhere you want on the screen.

Notice also that this dialog box is divided into two main sets of options, as indicated by the tabs: Layout and Motion. Feel free to click on each now just to get familiar with the look and feel of each part.

There are four things we need to do at this point: 1) change to the motion type that allows for data-driven motion in two dimensions; 2) identify for Authorware which of the three displays we want to animate; 3) show Authorware in which part of the screen we want the animation to take place; and 4) give Authorware two pieces of data, one for the horizontal position and one for the vertical position of the animation. We will need to toggle between Layout and Motion to accomplish these four tasks.

☐ **Click on the "Layout tab" to view the layout attributes** (if this is not already in view)**.**

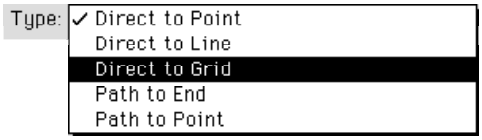
The Motion icon's dialog changes to the following:



Wow! Looks pretty intimidating doesn't it? Don't be put off by the apparent complexity of this dialog box. As you will see, it is very logical and procedural.

First, we need to change the motion type from "Direct to Point" to "Direct to Grid."

☐ **Click and hold on the words "Direct to Point" to activate the pop-up menu, then select "Direct to Grid":**

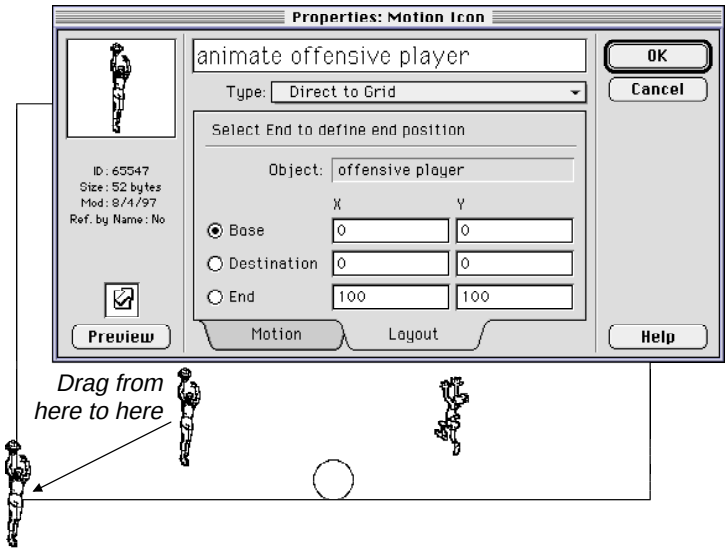


The other two — "Direct to Point" and "Path to End" are used to create animations that are fixed, that is, they do not change once set. However, starting with Authorware 4.0, the "Direct to Point" type also has limited data-driven characteristics (screen locations, in pixels, can be designated with variables).

This is one of three main data-driven motion types available in Authorware (the other two being "Direct to Line" and "Path to Point."). The point is that Authorware will calculate the motion of the animation based on the data provided.

Notice that Authorware is giving us directions as to what to do next just below the motion type: "Drag object to base position."

☐ **Drag the offensive player to the bottom left corner of the basketball court.**



Again, move the dialog box to a convenient spot on the screen for this and the remaining steps.

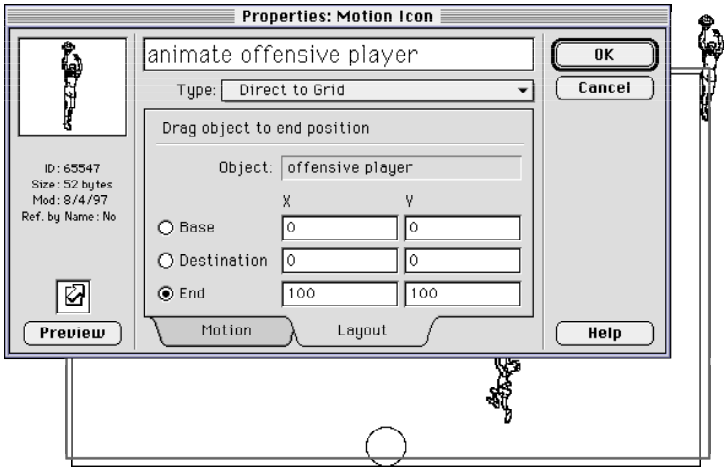
Believe it or not, Authorware now knows two important things. First, it knows which display is to be animated. Second, it knows the "base position" of the area in which the animation will take place. We now need to tell Authorware where to put the "end position" of the animation — we do this by first clicking in the radio button beside "end" and then dragging the offensive player to the opposite corner of the basketball court. (Notice that the directions in the dialog box have changed to "Select End to define position.")

☐ **Click on the radio button beside "end."**

Notice the directions in the dialog box have changed.

☐ **Now drag the offensive player to the top right corner of the basketball court.**

Your screen should resemble the following:



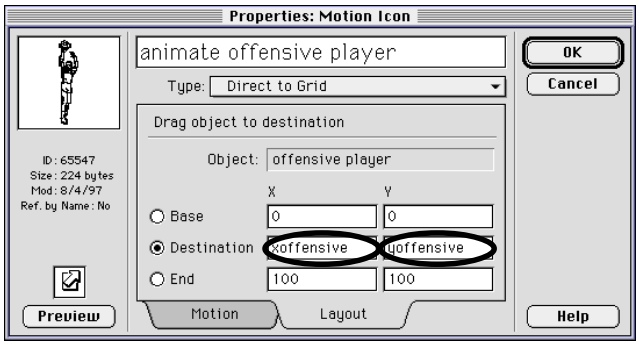
Notice that a fuzzy rectangle appears. This rectangle defines the 2-dimensional area in which the animation will take place. You can modify this area simply by moving the offensive player to a different spot (be sure to click on either the "base" or "end" radio buttons to tell Authorware which part of the space you wish to change).

Now you can see why we needed the basketball court to appear. It acts as the perfect guide. In fact, do your best to fine tune the position of the fuzzy rectangle so that it exactly matches the rectangle of the basketball court.

The last step is to tell Authorware about the two pieces of data it should use to control the animation.

☐ **Type "xoffensive" and "yoffensive" in the two text boxes as shown here:**

Make sure that you are viewing the Layout options (if not, click once on the Layout tab).



Of course, we could just type two numbers here, such as "12" and "68." But that would mean that the offensive player would always go to the same spot every time. Our goal is to have this spot **vary**, hence the need for a variable.

Notice the 0s and 100s in the boxes beside "Base" and "End." These numbers tell Authorware the numerical dimensions of the animation space, where X is the horizontal dimension and Y is the vertical dimension. Unless we change it, the base position (lower left hand corner) has a value of 0 both horizontally and vertically and the end position (top right hand corner) has a value of 100 both horizontally and vertically. These seem like reasonable dimensions, so we'll keep them. Obviously, you have much control and flexibility in how you want to set up this animation space (in fact, any of these dimensions could also be controlled by variables).

Time to test the animation.

☐ **Click "Preview."**

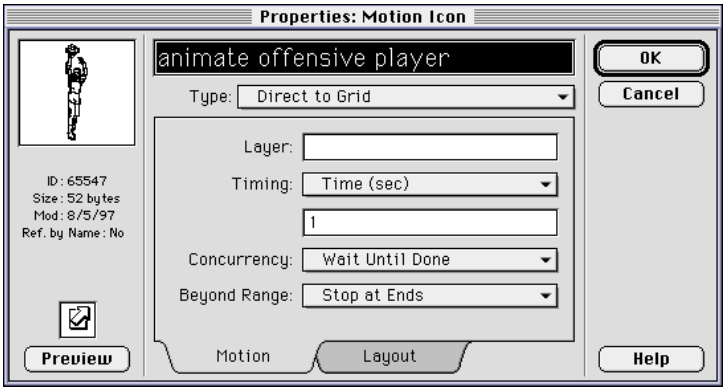
The offensive player animates from its original location to the spot "80 over" and "80 up." (Notice that two 80s now appear in the boxes under "Current." Why 80? Well, recall that we assigned the value 80 to the two variables in the Calculation icon. Authorware simply checks the current value of these variables and uses that information to animate the object.

You may need to move the dialog box way over to the right, barely exposing the Preview button, in order to see the complete animation.

There are lots of other choices that can be made in this dialog box. For example, let's take a look at the Motion attributes of this animation.

☐ **Click the "Motion" tab to view the motion attributes:**

Get used to toggling between the different parts of Authorware dialog boxes.



As is, the animation will take 1 second to complete. "Wait until done" means that any icons on the flow line that come next will not be executed until this animation is completed. "Stop at ends" means that the object will never go beyond the edges of the animation area. All interesting things to consider, but let's not make any changes at this point.

☐ **Click "OK."**

☐ **Select "Quit" from the file menu to go back to the flow line.**

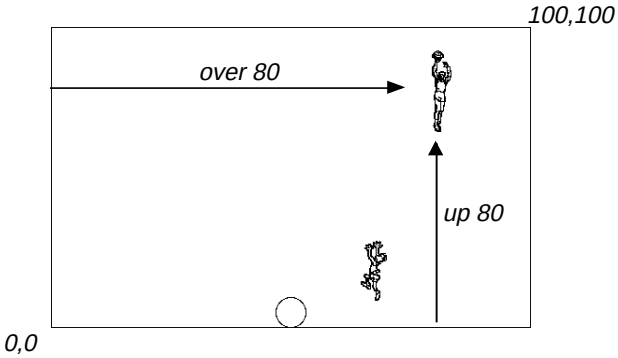
More math symbolism! When you see two numbers like this separated by a comma (100,100), it represents a screen coordinate where the first number is the horizontal position and the second number is the vertical position. (Mathematicians use the letter "x" for the horizontal axis and "y" for the vertical axis.)

You have successfully defined the animation for the offensive player.

Let's see if the animation is working.

☐ **Select "Restart" from the "Control" menu.**

The offensive player should animate smoothly from its original location to the top left corner of the basketball court, in other words, to the coordinate (80,80).



Press Command/ Control-R several times to view the animation repeatedly.

Of course, it goes to this screen location *every time* you restart the lesson since the Calculation icon, as of now, *always* assigns the values of 80 to the two variables.

Let's change it so that it will move to a random location on the screen.

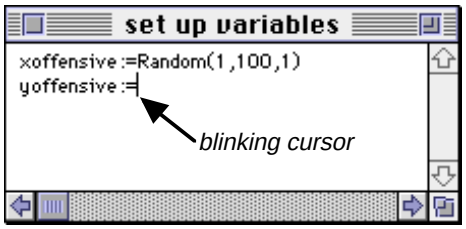
☐ **Go back to the flow line by selecting "Quit" from the file manu**

= set up variables

☐ **Double-click on the Calculation icon titled "set up variables."**

We want to replace the 80 on both lines with a function that selects a random number. This can be done one of two ways. You can either just type the function (assuming you've memorized the command), or you can ask Authorware to paste the function for you. Let's do it both ways.

☐ **Replace the "80" on the first line with "Random(1,100,1)" and delete the "80" on the second line, leaving the blinking cursor at the end:**



This function tells Authorware to choose a random number from 1 to 100 in increments of 1. Authorware then assigns this number to the variable "xoffensive."

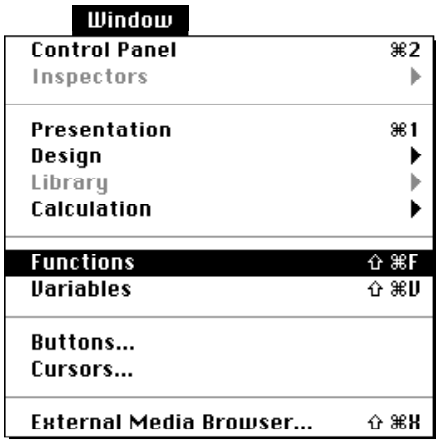
We will now use this same function with the variable "yoffensive." Instead of typing it, let's have Authorware paste it in for us.

☐ **Click once at the end of "yoffensive:=" and make sure the cursor is blinking at that point.**

This is important because Authorware pastes functions at the text insertion point.

☐ **Select "Functions" from the "Window" menu:**

A shortcut is to click on the function button on the tool bar

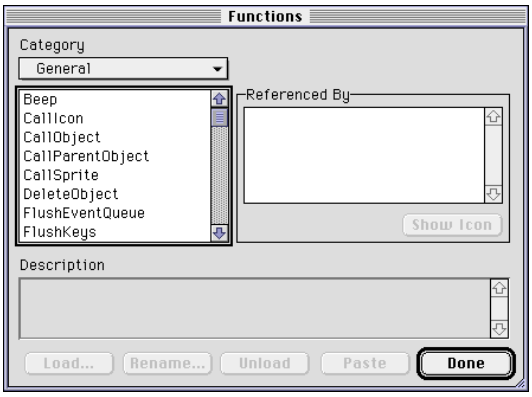


This is typed without any spaces, though extra spaces generally don't cause any problem.

Some particularly alert readers may have noticed that a colon (:) has "magically" appeared before the equal sign. This was inserted by Authorware to signify this is an "assignment" of a value to the variable as compared to an algebraic expression (used when determining whether the expression is true or false).

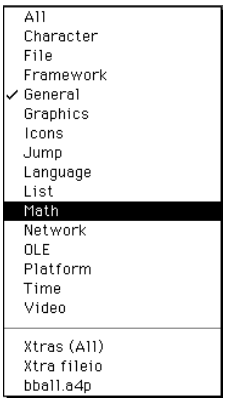
Programming purists will probably insist that you should type the colon from the start to demonstrate your understanding. However, this is one of the few areas in which I remain shamelessly lazy to this day — I never type the colon. Instead, I let Authorware do it for me.

This is the Functions dialog box:



I strongly recommend that you take some time at this point to browse over the categories of functions. Start getting familiar with them now — it will save you time later.

From here, you can access dozens of built-in Authorware functions (you can even add others). Notice that we are currently viewing the general category of functions. The function we want — random — is a math function.



☐ **Click and hold on the word "General."**

Macintosh Alert!

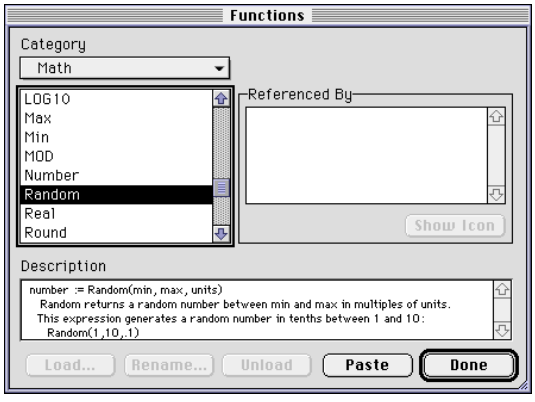
MacOS 8 users do not need to master "click and hold" because of the "sticky menu" feature of MacOS 8.

This activates a "pop-up" menu of the other categories of functions.

☐ **Continue to hold the mouse button down while you move the pointer to the word "Math," then release the mouse button.**

You are now viewing all of the math functions.

☐ **Scroll down the list of functions until the word "Random" appears, then click once on it:**



Notice that Authorware gives you a good description of the function.

☐ **Click “Paste.”**

This pastes the random function into the Calculation icon’s open window at the text insertion point. That’s all we need, so we are done.

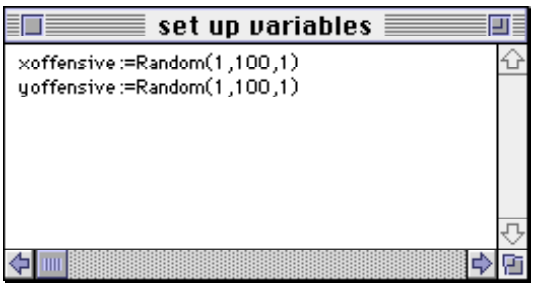
☐ **Click “Done.”**

Notice that Authorware pasted the function with the words “min,” “max,” and “units”:

```
yoffensive:=Random(min, max, units)
```

Obviously, we need to replace these with values of our choice.

☐ **Replace “min” with 1, replace “max” with 100, and replace “units” with 1.**



Although it would have been easier to have just typed in the random function for the second line, you now see how you can simply peruse the list of functions, find the one you want, then paste it in at any text insertion point. This will save you from having to consult the manual constantly to find useful functions.

☐ **Click on the close box in the Calculation icon’s window.**

☐ **Click “Yes” when asked if you want to save calculation changes.**

You should now be back at the flow line. (If you typed something in error, or if you forgot to replace something, you may have been given an error message or a “new variable” dialog box. Just click “cancel” or “continue” to go back to the Calculation icon window to fix the problem.)

Let’s see if this function is working properly.

☐ **Select “Restart” from the “Control” menu (or press Command/Control-R) .**

The offensive player should animate to a random spot in the animation area.

Keep testing it by restarting the file several times in a row.

☐ **Press Command/Control-R two or three times, watching the animation each time.**

☐ **Press Command/Control-J to jump back to the flow line.**

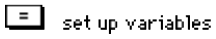
Have you saved lately? Let's do it now just to be sure.

☐ **Select "Save" from the "File" menu.**

Setting up the data-driven animation for the defensive player

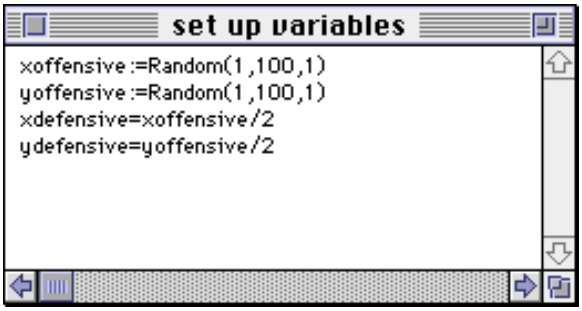
The next step is to set up a similar animation sequence for the defensive player. The steps are virtually the same as that followed to set up the animation for the offensive player: creating a unique set of variables that will control the animation for this object, then creating the animation with the Motion icon. One difference will be the mathematical model that controls its motion.

Creating variables



☐ **At the flow line, double-click on the Calculation icon titled "set up variables."**

☐ **Type the last 2 lines inside the window:**



Remember, Authorware will automatically "convert" the equal sign into the assignment operator (:=) — you can type either.

As you can see, we are creating two new variables, one to control the horizontal position of the defensive player ("xdefensive") and one to control the vertical position ("ydefensive"). Remember, the idea of this simulation is to have the defensive player go exactly to the halfway point between the offensive player and the goal, hence the reason to divide the offensive variables by 2. (You'll see later that there is a minor flaw in this reasoning.)

So, the third line tells Authorware to make "xdefensive" equal to "xoffensive" divided by two. The fourth line tells Authorware to make "ydefensive" equal to "yoffensive" divided by two.

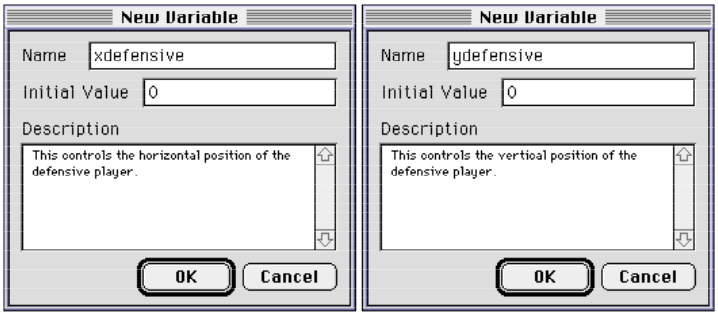
☐ **When finished typing these two lines, click on the close box in the Calculation icon's window.**

☐ **Click "Yes" when asked if you want to save calculation changes.**

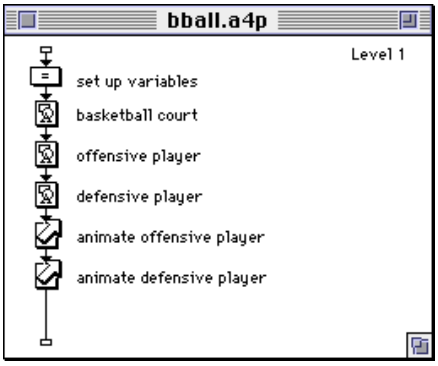
Since this Calculation icon contains two new variables, Authorware will display "new variable" dialog boxes. Like before, you should enter an initial value plus an appropriate description (though neither is absolutely required).

☐ **At the "New Variable" dialog box, enter an initial value of 0 and type an appropriate description for the variable "xdefensive"; Click "OK".**

☐ **At the "New Variable" dialog box, enter an initial value of 0 and type an appropriate description for the variable "ydefensive"; Click "OK".**

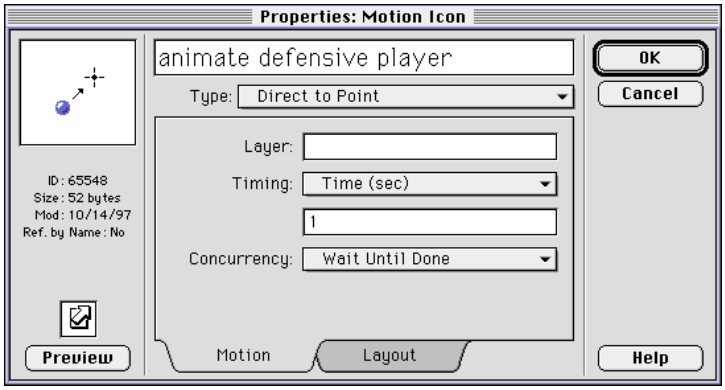


☐ **Drag a Motion icon to the bottom of the flow line and title it "animate defensive player."**



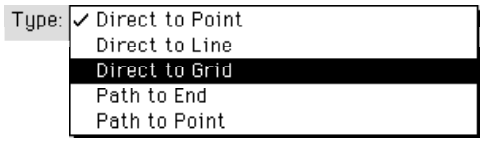
☐ **Select "Restart" from the "Control" menu.**

Again, Authorware will pause when it encounters an undefined icon. The dialog box for the Motion icon appears:



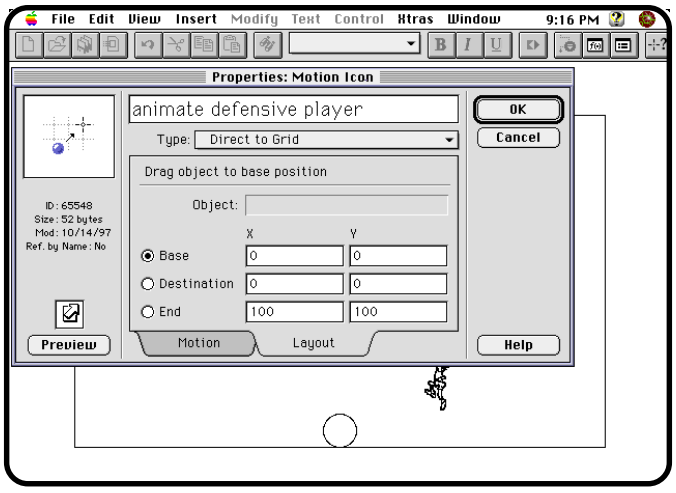
We again need to change the motion type from "Direct to Point" to "Direct to Grid."

☐ **Click and hold on the words "Direct to Point" to activate the pop-up menu, then select "Direct to Grid":**



☐ **Click on the Layout tab to go to the layout options.**

Your screen should resemble the following:



☐ **Drag the defensive player to the bottom left corner of the basketball court.**

Again, you will almost certainly need to move this large dialog box out of the way several times during these steps.

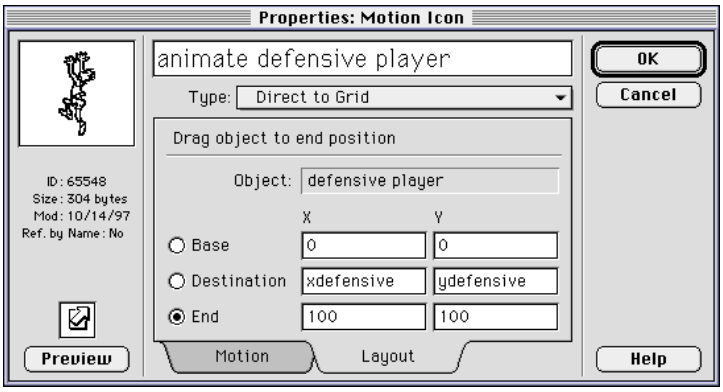
☐ **Click on the radio button beside “end.”**

☐ **Now drag the defensive player to the top right corner of the basketball court.**

☐ **Fine tune the position of the animation area (fuzzy rectangle) so that it aligns perfectly with the basketball court floor.**

Remember, click alternately on the radio buttons beside “Base” and “End” to fine tune each respective corner.

☐ **Type “xdefensive” and “ydefensive” in the two text boxes beside “destination”:**

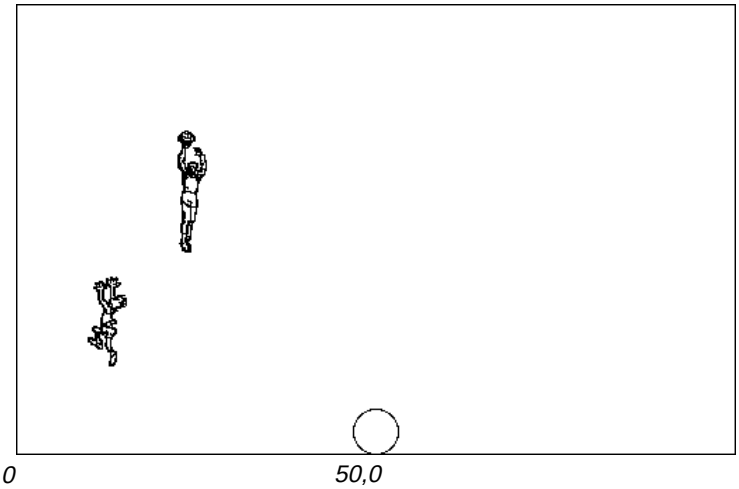


☐ **Click “OK.”**

☐ **The defensive player will animate to its position on the basketball court.**

☐ **Restart the file several times by pressing Command/Control-R.**

Watch the animation. You’ll probably get something like this (consider this a “snapshot” in time):



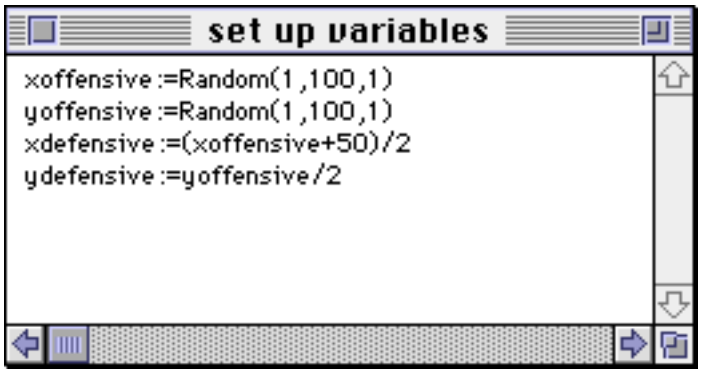
Something seems to be wrong. The idea was for the defensive player to be exactly half way between the offensive player and the basket. Actually, the animation is working just fine. The defensive player is going exactly half way, but the point of origin is still the bottom left corner of the basketball court. The "ydefensive" variable needs no adjustment, but the "xdefensive" variables needs to reflect the fact that the hoop is installed at the 50 mark on our number line.

Although there are many solutions to this problem, we will modify the "formula" for the "xdefensive" variable in the Calculation icon "set up variables."

☐ **Select "Quit" from the "File" menu to go back to the flow line.**

☐ **Double-click on the Calculation icon titled "set up variables."**

☐ **Edit the third line as per the following:**



This change moves everything to the right by 50 units. The parentheses are important. In a sense, they "force" Authorware to do the calculation inside them first and only then divide the whole thing by two.

You will find parentheses to be very useful symbols in programming. You are encouraged to make liberal use of them. You never have to worry about overusing parentheses — Authorware will ignore them if they are redundant. There's only one important rule to remember: "For every left parenthesis, there must be a right parenthesis." (Don't worry, Authorware will remind you if you forget.)

☐ **Click on the close box in the Calculation icon's window.**

☐ **Click "Yes" when asked if you want to save calculation changes.**

☐ **Restart your file by pressing Command/Control-R.**

Your animation should now work properly. The defensive player should move to the proper position halfway between the offensive player and the basket. Restart your file repeatedly to make sure it is working.

☐ **Press Command/Control-J to return to the flow line.**

Let's save our file.

☐ **Select "Save" from the "File" menu.**

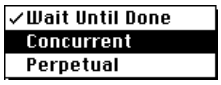
Concurrent animation

Let's make one small change to enhance the animation. As is, the defensive player waits for the offensive player to get into position before moving. This is not a sound basketball strategy! Instead, the defensive player should move with the offensive player. This is easily done by changing the offensive player's motion to **concurrent** animation.

☐ **Double-click on the Motion icon titled "animate offensive player."**

The "Concurrency" attribute is found in the motion part of this dialog box.

☐ **Click on the "Motion" tab.**



☐ **Click and hold on the phrase "Wait Until Done."**

This activates another "pop-up" menu.

☐ **Continue to hold the mouse button down while you move the pointer to the word "Concurrent," then release the mouse button.**

Be sure to only change the motion to "Concurrent" for the offensive player. Leave the defensive player's motion set to "Wait Until Done."

☐ **Click "OK."**

☐ **Press Command/Control-R to restart your file.**

Now, the defensive player animates at the same time as the offensive player each time you restart your file.

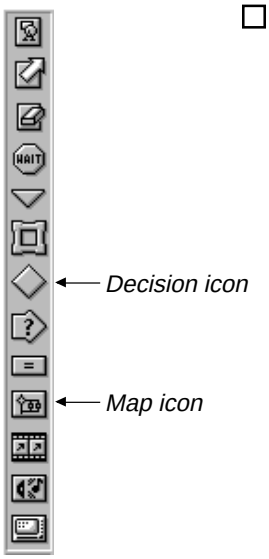
☐ **Press Command/Control-R to restart your file.**

☐ **Press Command/Control-J to jump back to the flow line.**

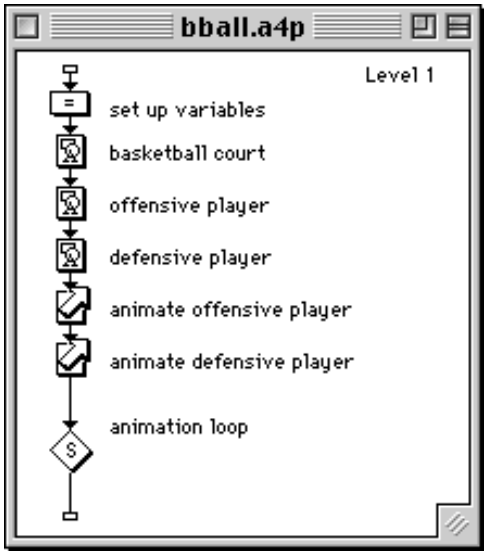
☐ **Select "Save" from the "File" menu.**

Making the animation run continuously in a loop

There is one final thing we will do in this session. Up to now, we have had to continually press Command/Control-R to restart the file. Let's change the file to run continuously — to repeat until we click the mouse or press a key to stop it.



☐ **Drag a Decision icon to the flow line and title it "animation loop":**

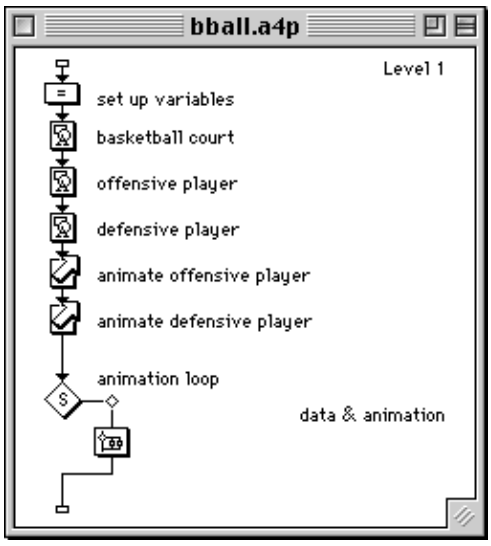


We need to put the following three icons in the loop: "set up variables," "animate offensive player," and "animate defensive player." The icons need to be executed in that order. However, you can only attach single icons to the Decision icon, so we will attach an empty Map icon to the Decision icon first. Then we'll put the three icons inside of it.

☐ **Drag a Map Icon to a point just to the right of the Decision icon and release the mouse button.**

☐ **Title this Map icon as "data & animation."**

It should "attach itself" to the Decision icon as shown here:



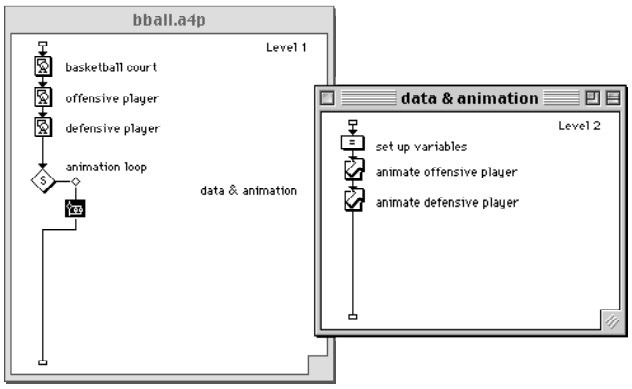
Next, we will move the Calculation icon and two Motion icons *into* the Map icon.

☐ **Double-click on the Map icon "data & animation."**

Each Map icon contains its own flow line which is a continuation of the flow line entering the Map icon. As the flow line exits the Map icon, it "connects" back to the first flow line. To distinguish the flow lines, Authorware labels this one as Level 2 (up to this point we working only on the Level 1 flow line). (If we were to put a Map icon inside a Map icon, it would have a designation as Level 3.)

The next step is to drag the three icons into the level 2 window of the Map icon. (You may need to adjust the size and location of your two windows first.)

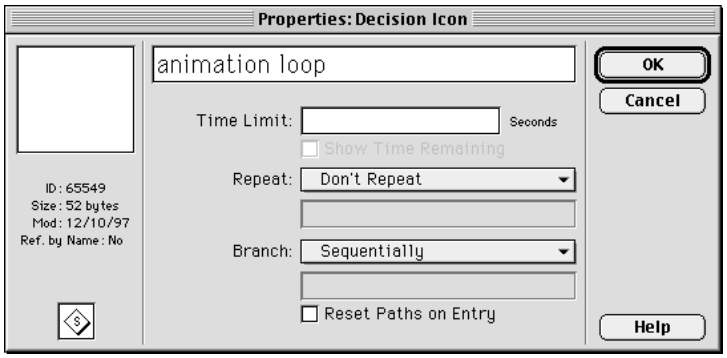
☐ **Drag the Calculation icon "set up variables" into the Map icon's level 2 window, followed by the icons "animate offensive player" and "animate defensive player."**



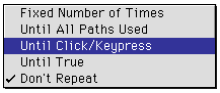
It's easy to get confused at first about the flow line levels. However, they are analogous to folders or directories on the computer's desktop. Levels are not unique names to the flow lines inside the Map icons, but merely refer to relative "layers" — you can have dozens of level 2 and level 3 flow lines. There is really no limit to how "deeply" you can nest Map icons inside each other. Map icons are powerful ways to organize your files.

Next, we need to set up the loop.

☐ **Double-click on the Decision icon "animation loop."**



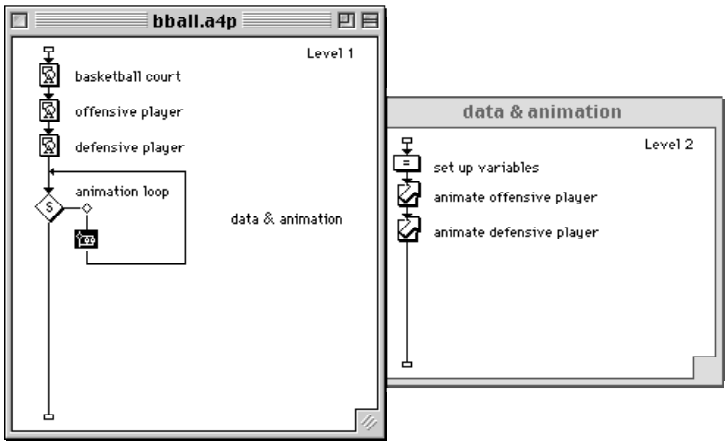
Decision icons have two main settings: repeating and branching. In this example, we only have one branch — to the Map icon "data & animation" — so we will leave that set to "Sequentially." However, we need to change the Decision icon to repeat "until click/keypress."



☐ **Click and hold on the words "Don't Repeat" to activate the pop-up menu, then select "Until Click/Keypress".**

☐ **Click "OK."**

Your flow line should now resemble this:



The flow line visually represents the loop.

Of course, you can close and open the level 2 window as you wish — it doesn't matter to the way Authorware executes the file.

☐ **Select "Restart" from the "Control" menu.**

Your simulation should run smoothly. As the offensive player moves to random locations on the basketball court, the de-

fensive player should likewise move to exactly to the midpoint between the ball and the hoop. Enjoy!

"Exactly?" Well, probably not. Unless the two animation areas defined in the two Motion icons are positioned exactly over the basketball court floor, there is probably an error of at least 1-3 pixels.

☐ **Click the mouse button or press any key to stop the simulation.**

Press Command/Control-J to go back to the flow line.

Let's save it one more time.

☐ **Select "Save" from the "File" menu.**

☐ **Package the file.**

See Appendix A for background and instructions on packaging.

Summary

You animated two displays according to data generated by the computer and stored in variables. You've explored how to create and use variables. You've also seen how to use one of the many Authorware functions (i.e. random number generator). The animation of the defensive player was governed by a simple mathematical rule that we programmed into the computer. As a result, you've created your first simulation!

But is it interactive?

Although this little example does a nice job of introducing how to use variables and functions to control data-driven animation in a simulation, it hardly qualifies as "interactive." Let's face it, all you do is restart the file and then watch. An example of a modification that would truly make this interactive would be to put the offensive player's position under the control of the user such as by dragging the offensive player from point to point with the mouse. How to make this modification is described in Chapter 6. A word of warning, however. This requires the use of a not so obvious programming strategy that may not make much sense at this point.

Other projects

1. Change the "mathematical engine" so that the defensive player moves one third or one fourth of the distance between the offensive player and the hoop.

2. Change the scenario (or context) from basketball to another sport of your choice, such as tennis or soccer just by editing the Display icons and swapping or changing the graphics. Better yet, change the offensive player to

Darth Vader and the defensive player to Luke Skywalker protecting R2D2! Invent another simple rule for the animation to follow.

Don't forget to get copyright permission before you go commercial!

3. Instead of having the offensive player go to a random number, have it follow a rule that you design. For example, have it start at position (80,100), (i.e. over 80 and up 100) but then subtract 5 from the variable "yoffensive" during each animation loop to have the player "walk" down the court.

Notes:

Chapter 3: Mystery Number

Building a simple game with data-driven animation

This chapter shows how to create a simple math game that uses data-driven animation.

In this session you will be saving the following file:

mystery.a4p

This chapter assumes you are familiar with the structure of Authorware files, the use of the flow line, and the Display, Map, and Calculation icons. It also assumes that you have an introductory knowledge of how to create and use variables and system functions, similar to what was presented in Chapter 2.

The Problem

You've been volunteering your time at the local elementary school. The students in the fifth grade class don't seem to have any interest in mathematics. The only math experiences they have had have been with worksheets and drills. They also seem to lack important understanding of some basic number theory concepts. You would like to give them a fun and interesting math game to experience. The classroom has several older computers (with 12" RGB monitors), but the teacher doesn't have any good math software for them. You want the game to be entertaining, but you don't want it to merely "sugar coat" the content. Instead, you want a math adventure that weaves mathematics into the game context. You want the students to *use* mathematics as they play the game. Maybe Authorware can help.

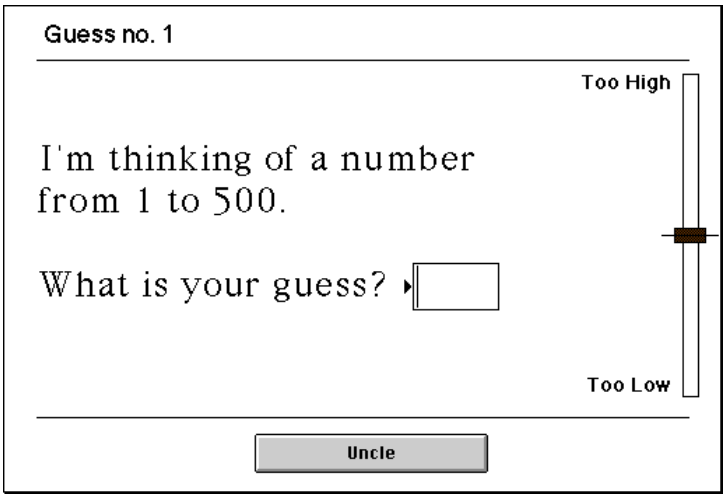
Introduction

Last chapter, you saw how to create variables to control the animation of screen objects. This chapter uses the same techniques, but in a game format involving intense user interaction. Whereas the previous chapter showed you how to create a pure simulation (with no instructional overlay), this chapter introduces you to game design. The programming is similar in that it will be important to design and construct a "mathematical engine" that controls not only the game elements, but also aspects of the interface, such as giving the user adequate feedback.

The goal of the game is to determine the identity of a "mystery number," that is, a number chosen randomly by the computer. The game will give players as many guesses as they wish, with the number of guesses acting as a score. The game will also give players the chance to give up and start over.

One of the most important aspects of the game is the way in which it tells players the "correctness" of their guess. There are lots of possible ways of doing this. You've probably played different versions of this game, such as one person thinking of a number and then telling the other person if their guesses are "too high" or "too low." Another is the "hot/cold" version where you tell someone if their guesses are getting closer ("You're getting warmer!") or if the guesses are far off the mark ("You're freezing!")

We will merge both forms of the game here. Here's a snapshot of the main screen of the game:



As you can see, we will be giving the user as many guesses as they want with the computer keeping track of the total number used. The "Uncle" button will be a way for the user to give up and to have the computer tell them the answer. Most important is the graphic to the right of the screen. A pointer or guide will slide up and down on the scale indicat-

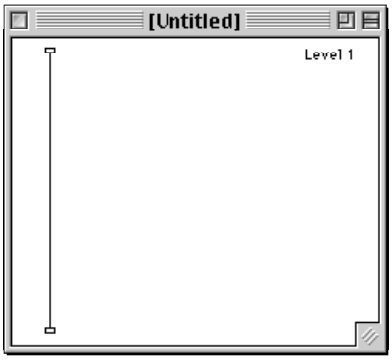
ing not only if the guess is too high or low, but how close the guess is to the answer (located at the very center of the scale). Guesses closer to the center are "hotter" than guesses toward either edge of the scale.

Getting started

The first step is to launch the Authorware application.

☐ **Start Authorware by double-clicking on the Authorware application.**

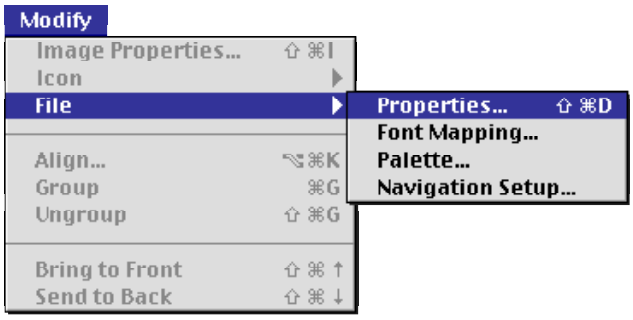
A new file opens with a Level 1 flowchart with the name "untitled":



Setting up the file structure

From the description of the problem, we know that the classroom only has some older computers with 12" RGB monitors. Let's change the size of the presentation window right away to match these specifications.

☐ **Select "File" under "Modify" from the menubar, then select "Properties...".**



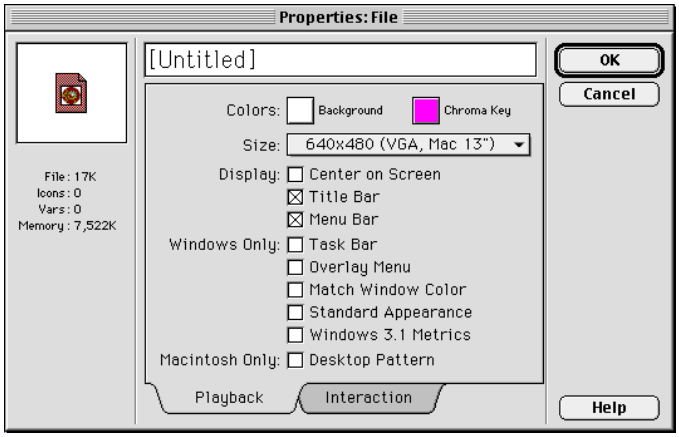
This is a good place to use some cultural conventions to improve the feedback. Red is generally associated with "hotness" and blue with "coldness." Therefore, you might fill the scale with a gradient going from dark blue to dark red from the edges of the scale to the center.

Cultural conventions such as the use of color to represent temperature are very interesting. Yes, burning objects, like wood, can be "red hot," but "white hot" is even hotter even though we don't normally associate the color white with heat. Do things really turn blue when they get cold? What about blue flames in a fire? Yes, they are cooler than yellow flames, but they certainly won't freeze water.

Windows Alert!

This example was constructed on a Macintosh computer. Although your choices will be different, the procedure of how to change screen sizes is the same.

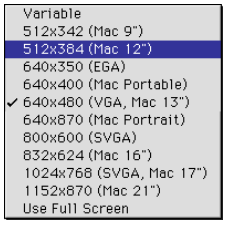
This opens the File Properties dialog box (be sure you are viewing the "Playback" options):



This is an important decision that needs to be carefully considered at the start because changing the screen size later means lots of extra work. Authorware will not resize displays for you if you change the screen size — you will have to do it yourself manually.

A good rule of thumb is to design for the lowest end hardware system you expect users to have.

As you can see, there are all sorts of important options we can change here. All we are interested in changing is the size of the presentation window.



☐ **Click and hold on "640x480 (VGA, Mac 13")" beside "Size".**

There are lots of choices. We want a size that will fit on a 12" screen.

Windows Alert!

You will have different presentation window choices. Choose a size smaller than VGA, such as Mac 9" (512x342).

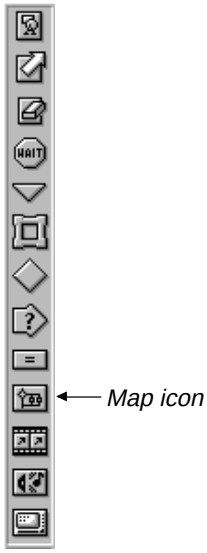
☐ **Select "512x384 (Mac 12")."**

512x384 (Mac 12") should now show in the box beside "Size".

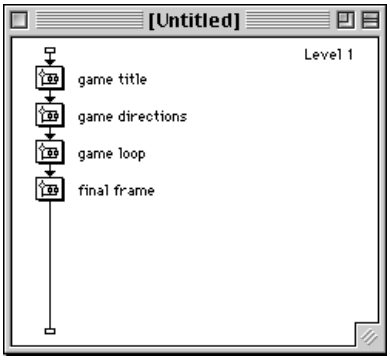
☐ **Click "OK."**

Setting up the game structure

A good way to start is by using Map icons to organize the overall structure of the main game parts.

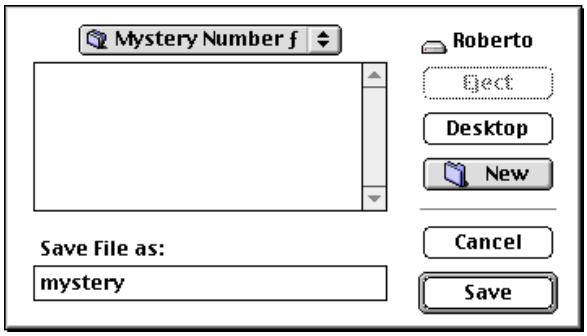


☐ **Drag four Map icons to the flow line and title them as per the following:**



Let's go ahead and save the file.

☐ **Select "Save as..." from the File menu.**

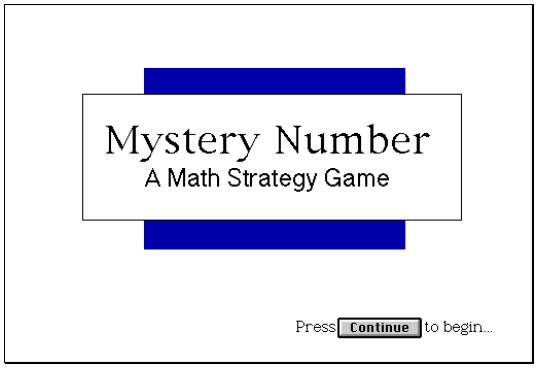


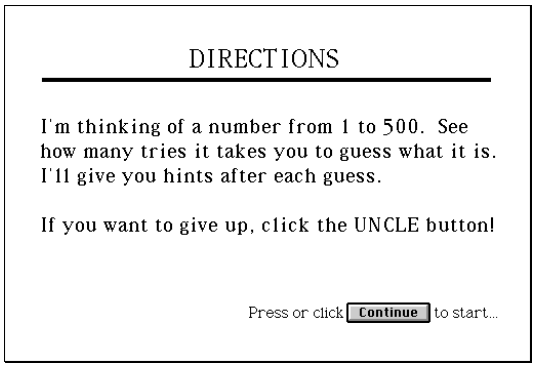
☐ **Save your file with the name "mystery" to the folder of your choice.**

Note again that Authorware will automatically add the extension ".a4p" to your file name.

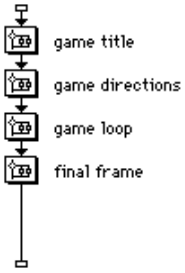
From here on, remember to save your file periodically using the "Save" command from the "File" menu.

We are not going to take the time to step through the design of a suitable title and game directions. Instead, we want to dive right into the heart of the game itself. However, here are simple examples of a title screen and game directions that you may want to incorporate later:





Writing game directions can be a fun yet very challenging language arts activity. In my work with children designing their own computer games, one of the unexpected positive outcomes was the intensity with which some of the children spent writing their game directions.



Let's start building the game itself.

☐ Double-click on the Map icon "game loop."

The first thing we will do is set up some of the variables that this game will need. Here's a description of all the variables that we will be using:

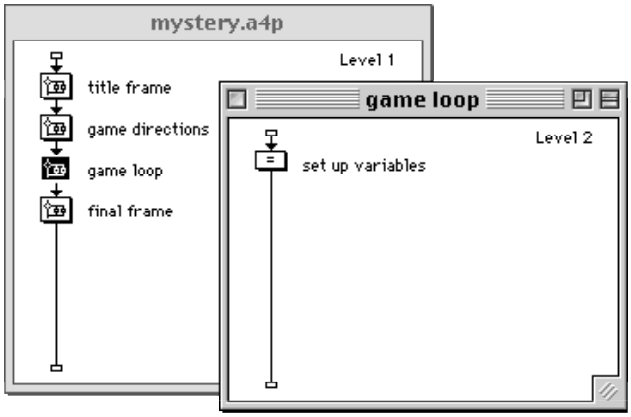
ANSWER - A user variable that selects a random number from 1 to 500 in multiples of 1.
GUESS - A user variable that keeps track of the total number of guesses.
NUMENTRY - A system variable that contains the number of the player's most recently entered guess.
DISTANCE - A user variable that computes the difference between the player's guess (NUMENTRY) and the randomly chosen ANSWER. A positive number indicates that the guess is too high and a negative number indicates that the guess is too low. This is also the variable that will generate the data needed by the animation.

All of these are numeric variables.

Although these descriptions might seem a little confusing right now, try to follow the logic behind them. Better yet, refer back to these descriptions as needed as you go through the rest of the chapter.

"User variables" are ones that you create and "System variables" are ones that are built into Authorware that you can access and use. Authorware is keeping track of all kinds of data for you whether you need them or not.

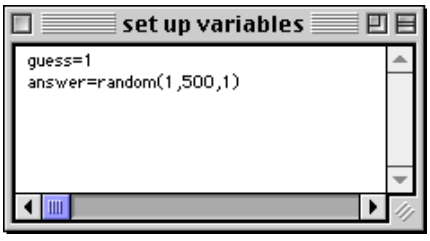
☐ Drag a Calculation icon to the Level 2 flow line and title it "set up variables."



At the start of each "game loop" we want Authorware to choose the random mystery number and also to reset the guess counter to 1.

☐ Double-click on the Calculation icon "set up variables."

Type in the following:



The first line sets the variable “guess” to 1 (the game always starts with the first guess). The second line uses the math function “Random” to set the variable “answer” to a random number from 1 to 500 in increments of 1.

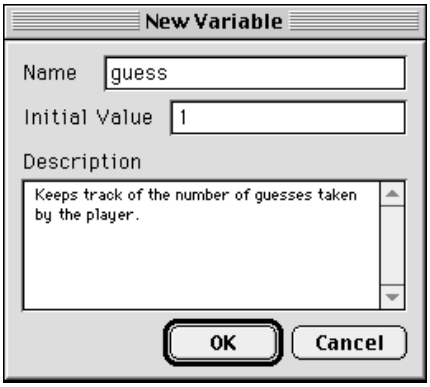
Remember that you can also choose to paste a function instead of typing it — refer back to chapter two for the procedure.

☐ **When finished, click on the close box in the Calculation icon’s window.**

☐ **Click on the “Yes” button to save the changes to this icon.**

Authorware notices that you have set up two new variables and asks you about each. A new variable dialog box automatically opens for the variable “guess.”

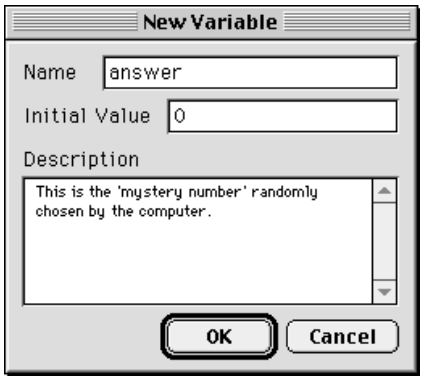
☐ **Give the variable “guess” an initial value of 1 and type a description similar to the following:**



☐ **Click “OK.”**

A new variable dialog box automatically opens for the variable “answer.”

☐ **Give the variable “answer” an initial value of 0 and give it a description similar to the following:**



☐ **Click “OK.”**

Building the interaction

Interaction icon

Now, let's set up the user interaction. This is accomplished with the amazing Interaction icon. This icon is the primary way in which Authorware allows you to set up ways for the user to interact or participate in your games. The better designed the user interface for interaction, the better the game.

An Interaction icon is part Display icon and part Decision icon (its shape comes from superimposing these two icons). Likewise, an Interaction icon has both display features and branching features.

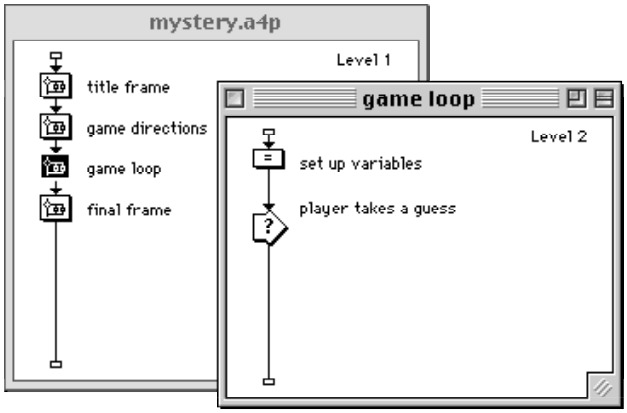
However, it adds the capability of getting and using input from the person controlling the keyboard or the mouse.

This icon also goes inside the game loop Map icon.

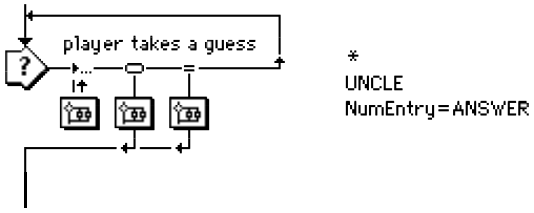
The range of features and options in this icon is remarkable. Almost every kind of user interaction you can think of can be constructed with it. Buttons, pull-down menus, movable and clickable objects, character entry, etc. can all be done with this icon — many with little or no programming.

However, the interaction we will be setting up requires much more "programming" than most beginners experience with Authorware.

☐ **Drag an Interaction icon to the Level 2 flow line just below the Calculation icon "set up variables" and title it "Player takes a guess."**



We will be attaching three branches to this Interaction icon. Here is a picture of what it will look like when we are finished:



The first branch, * (asterisk), will allow the user to type in a guess with the keyboard.

The second branch, Uncle, will give the user a button to press if they want to give up.

The third branch compares the user's guess to the right answer.

Look closely at the flow line. Notice that the second and third branches will exit the interaction (and consequently out of the "game loop" Map icon and into the "final frame" Map icon), whereas the first branch eventually brings the user back around to the interaction for another guess.

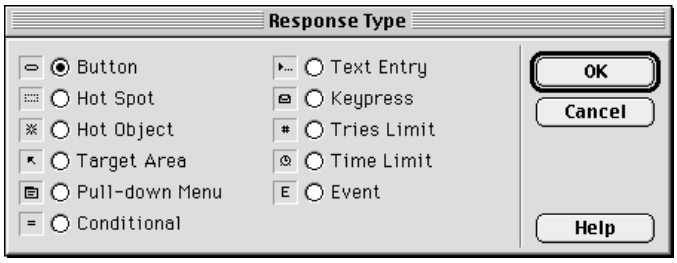
Also notice the graphic above each of the branches: three dots, an oval, and an equal sign. These denote different "response types." The three dots denote a text entry from the keyboard, the oval denotes a button, and the equal sign denotes a condition (or calculation).

OK, let's get started building this interaction.

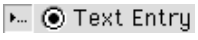
Setting up a text entry response branch

☐ **Drag a Map icon to the flow line, just to the right of the Interaction icon "player takes a guess."**

This dialog box always appears the very first time you attach any icon to an Interaction icon:



These are all of the possible response types. (Notice the graphical symbols to the left of the response type names.) This first branch will allow the user to type in their guess, so we need to change the response type from "Button" to "Text Entry."



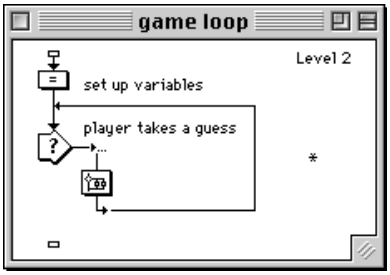
☐ **Click in the radio button beside "Text Entry."**

Despite its name, this response type also allows numeric entries.

☐ **Click "OK."**

☐ **Name this icon (or branch) "*" (an asterisk).**

Your flow line should look like the following:



Yes, an asterisk is a strange name for an icon title. Let's take a moment and explain what's going on. Up to this point, all icon names have been optional and arbitrary. We have tried to choose names that make sense to *us* not Authorware. Not so with text entry response branches. Authorware compares the icon's name with whatever the user types in. If it is a match, the flow of the file is "sucked into" this branch. If it is not a match, then the flow bypasses this branch completely. If we were asking a question at this point, such as "Who was the first president of the USA?" we might title this branch something like "Washington."

In our case, we need to anticipate the user typing in any number from 1 to 500. So, we use a special "wild card" character, the asterisk, that tells Authorware to match on *anything*.

Another special wild card character is the question mark (?). However, it only allows matches for a single character.

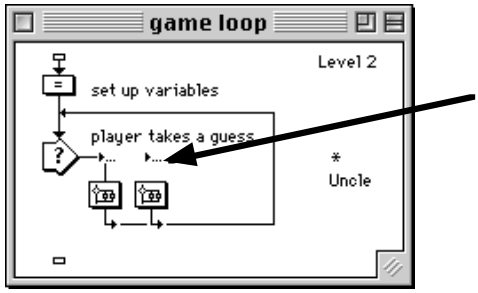
Setting up a push button response branch

Let's go ahead and set up the second branch that includes a pushbutton for the user to press if they want to give up.

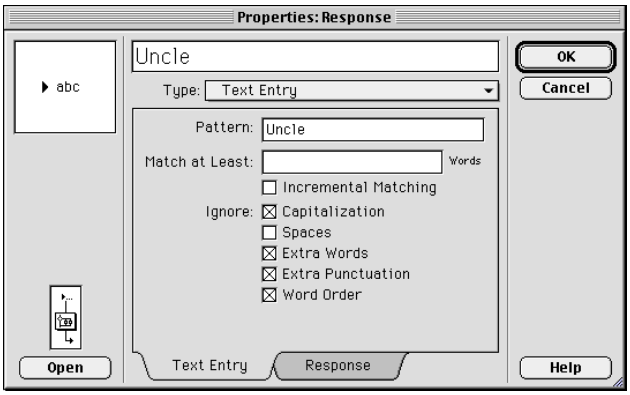
☐ Drag another Map icon just to the right of Map icon titled "*" and title it "Uncle."

Authorware assumes you want the same response type as the branch to the immediate left, so this one is also set as a text entry response type (as indicated by the three dots). We want to change this to a pushbutton, so we have to change the response type for this branch.

☐ Double-click on the graphic of the three dots leading into the branch "Uncle."

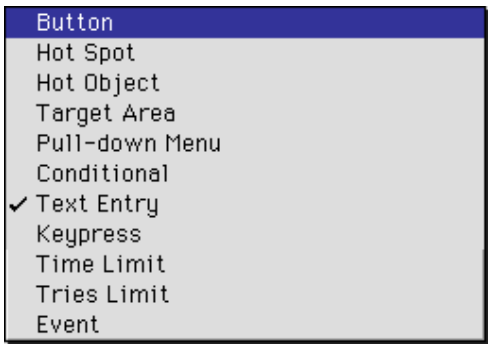


The following dialog box appears:

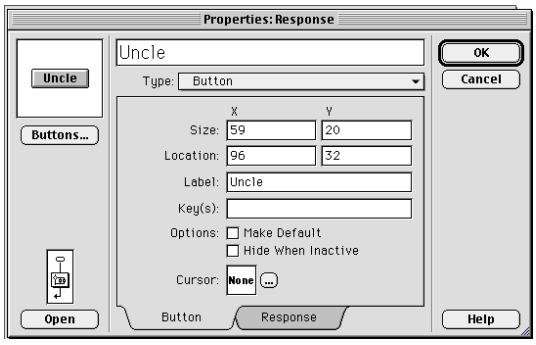


This contains options for the branch. The options are dependent on the response type of the branch (text entry in this case).

☐ **Click and hold on the words "Text Entry" beside "Type", then select "Button":**

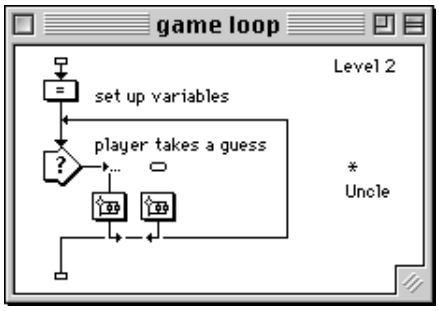


This was the same list of response types as seen earlier. The options in the dialog box now change to match those of a pushbutton response type:



☐ **Click "OK."**

You should now be back at the flow line. Notice that the graphic of the three dots has changed into a "baby pushbutton."

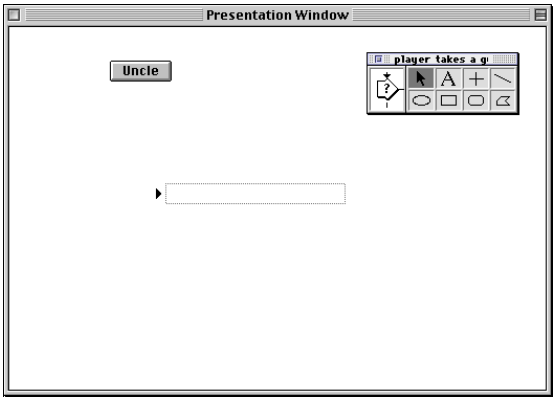


We'll set up the remaining two branches later. Let's shift our attention to the display characteristics of this interaction. (Remember, an Interaction icon has both display and branching capabilities.)

Setting up the Interaction icon's display

☐ **Double-click on the Interaction icon "player takes a guess."**

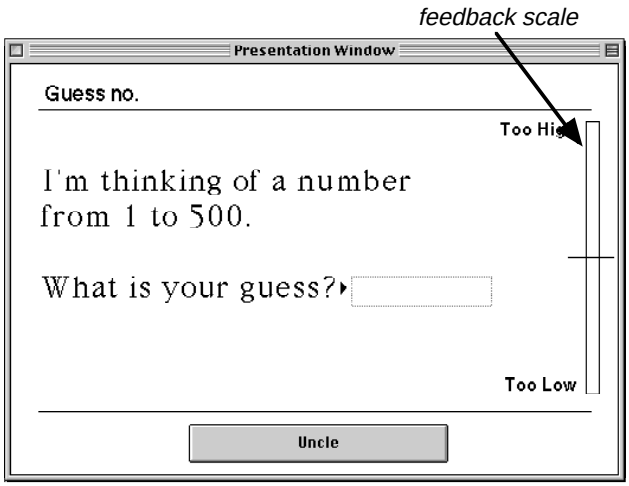
Authorware gives you a blank display window (along with a tool box) that's identical to what you get with a Display icon:

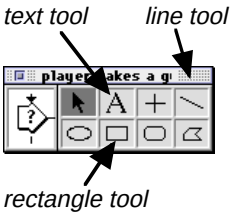


Well, almost identical. There is a small black triangle beside a fuzzy rectangle in the center and a button named "Uncle" up off to the side. These graphics were automatically generated by the two response branches. The small black triangle will point out to the user where to type in their guess. The fuzzy rectangle shows the size of the text entry area — it will not appear when we run the file. You can resize both the button and the text entry area by dragging any **corner**. You can move either just by dragging it to a new location.

Next, we need to construct the rest of the display.

☐ **Construct the display so that it resembles the following:**

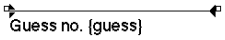




What's the best way to draw this little line exactly at the midpoint of the scale? One way is to turn on "Snap to grid" under "View" on the menubar and count the number of steps. Another trick is to draw this long rectangle as two short ones that overlap on one edge. Just draw one the way you want it, then copy and paste it. You can then align the end of the second short rectangle over the end of the first.

Besides moving and resizing the Uncle button and text entry area, you'll need to use the text tool, line tool, and rectangle tool to make this display. Be sure to draw the small horizontal line exactly at the midpoint on the feedback scale.

Showing embedded variables in displays



You may be wondering why we didn't go ahead and type a "1" type in beside "Guess no." That's because we want Authorware to do the work here. Recall that we set up a numeric variable called "guess." We need to embed this variable in the display so that Authorware will show its current value throughout the game. There are two ways to do it. If you know the name of the variable, you can just type it in within a set of braces, such as "{guess}." The braces alert Authorware that the thing inside is a variable — they will not be displayed when the file is run. The other way is to have Authorware paste the variable in for you at the text insertion point — handy when you don't remember the variable name exactly. Let's do it the second way just for practice.

I like to refer to braces as "fancy brackets".

This is similar to the way one pastes functions — see chapter 2 for a review.

☐ **Choose the text tool, then click just to the right of "Guess no." in the display window.**

Be sure to type a space at the end!

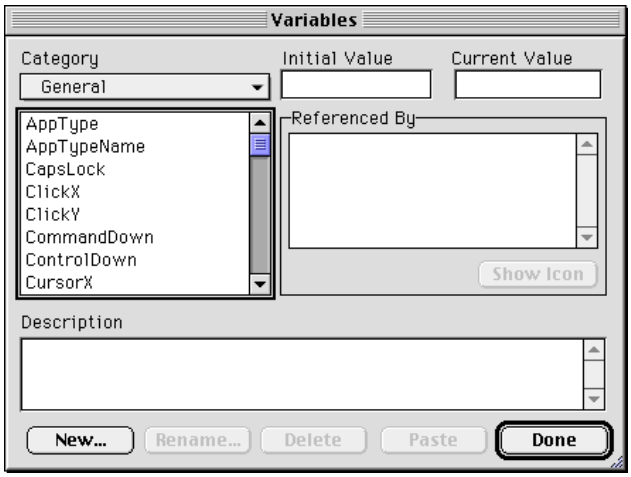
The text window will open with the cursor blinking at the text entry point.

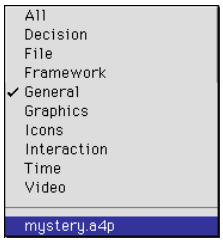
☐ **With the cursor blinking, select "Variables" from the "Window" menu.**

Toolbar Shortcut!

Click here

This opens up the Variables dialog box with the category "General" most likely displayed:





☐ **Click and hold on "General."**

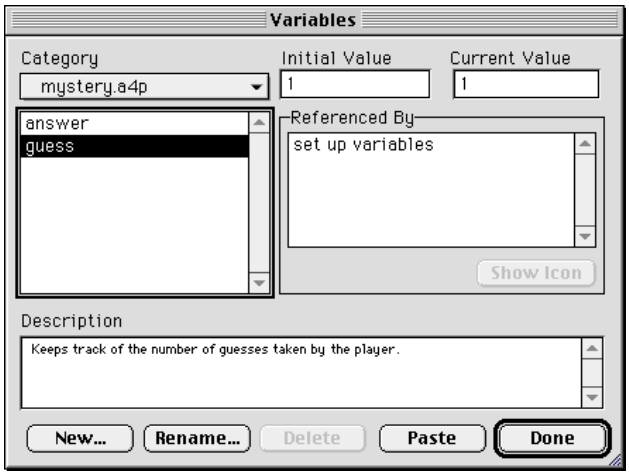
All of the Authorware variables are grouped by category (just like the functions). There are dozens of Authorware system variables. However, we want to find and use one of the variables that we have already set up — "Guess." Notice that there is a category at the bottom with the same name as the file. This is where all the user variables are kept.

Just like the functions, I strongly encourage you to browse through the many variables in each of the categories as soon as you can. It pays to get familiar with them as soon as you can.

☐ **Continue to hold the mouse button down while you move the pointer to the name "mystery.a4p," then release the mouse button.**

This shows all of the user variables in the file. So far, we only have two: guess and answer.

☐ **Click once on "guess."**



Here is where typing a good description earlier pays off.

Notice the information that pops up in the various fields.

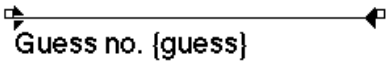
☐ **Click "Paste."**

This pastes the variable into the display at the text insertion point. That's all we need, so we are done.

Remember, this paste only works if the cursor is blinking at a text insertion point. You can't paste a variable onto the screen somewhere without first choosing the text tool and then opening a text window on the screen.

☐ **Click "Done."**

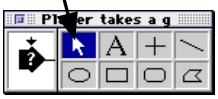
The result is the same as though you had typed it:



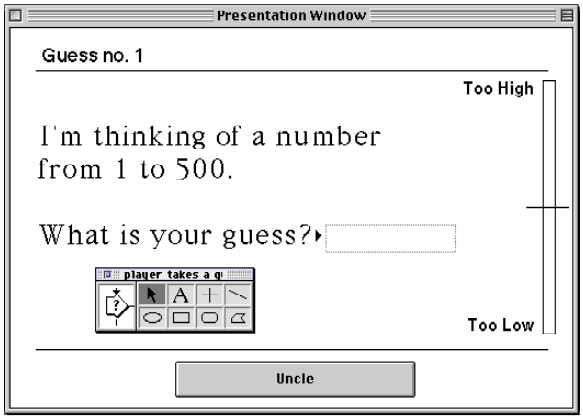
Let's see if it's working.

☐ **Click on the pointer in the tool box.**

Pointer



The variable name inside the braces should be replaced by the current value of the variable (probably 1). Your screen should now resemble the following:



☐ **When finished constructing the display, click in the close box in the tool box to go back to the flow line.**

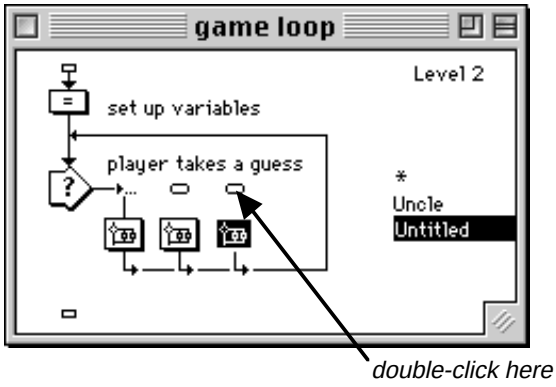
Adding the final branch

Let's finish constructing the remaining branch to the Interaction icon. The purpose of this branch is to determine if the user has correctly guessed the mystery number. In other words, to determine if a specific condition has been met.

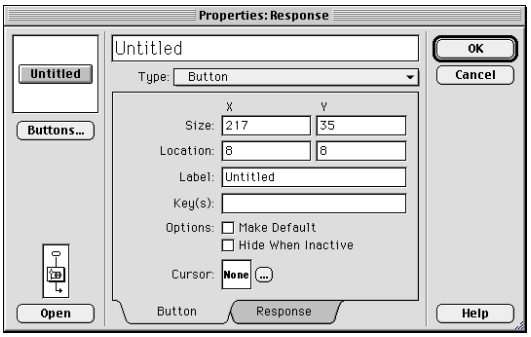
☐ **Drag a Map icon just to the right of the Map icon titled "Uncle."** (Leave it untitled.)

We need to change this branch to a conditional response type.

☐ **Double-click on the "baby pushbutton" graphic leading into the Map icon "Untitled."**

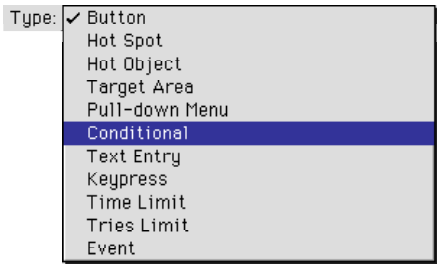


This brings up the options dialog box for this branch:



We need to change the response type from "button" to "conditional."

☐ **Change the response to "Conditional" by clicking and holding on the word "Button", then selecting "Conditional" from the pop-up menu:**

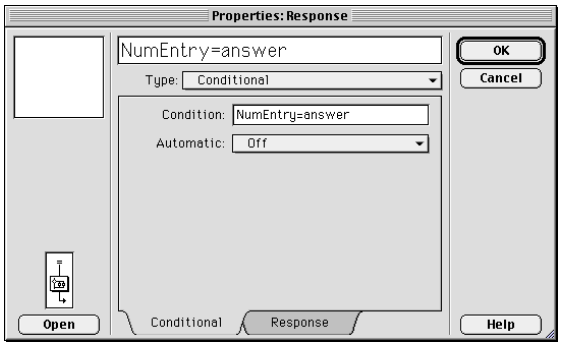


This changes the response type to "Conditional." The options dialog box changes accordingly.

This branch is meant to be triggered only on the condition that the player's guess is exactly equal to the random mystery number. Although this makes sense to us, Authorware is too dumb to understand "simple" English. So, the next step is to type in this condition in a way that Authorware understands. Fortunately, the player's guess is automatically recorded by Authorware at the text entry response branch (titled *) with the built-in system variable "NumEntry." Also, the mystery number is contained our variable "answer." The phrase "NumEntry=answer" does the trick.

Like all variables, you can always choose to have Authorware paste NumEntry at a text entry point. Select "Variables" under the "Window" menu. NumEntry is in the "Interaction" category. You will also find a good description of this variable there as well.

☐ **Type "NumEntry=answer" in the text field beside "Condition"** (be sure the "Conditional" options are shown)**:**



As you type, notice that this condition also automatically becomes the title for the icon.

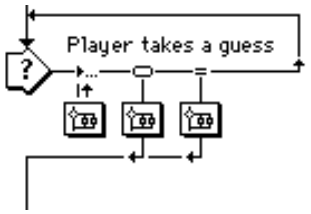
When Authorware encounters this response branch, in essence it asks itself whether or not the condition "NumEntry=answer" is true. If it is true, then it's a match and the flow of the file is sucked into this branch. If it is false, it bypasses the branch.

☐ **Click "OK."**

This sends us back to the flow line.

Changing the flow

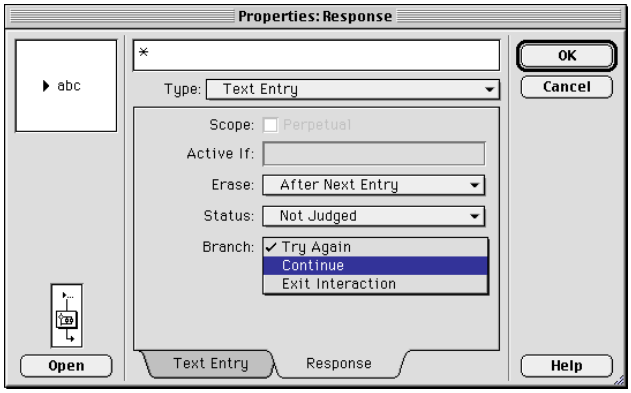
The next step is to change the direction of the flow of each of the branches to match the following:



*
Uncle
NumEntry=ANSWER

Changing the graphic of the flow is extremely important. Remember, the flow line is not just a pretty picture, but a visual representation of how Authorware actually executes your file (like any flowchart). Let's do it first, then we'll explain it.

Let's start by changing the flow out of the first text response branch titled "*". There are two ways to do it. First, we could open up the options dialog box for this branch (by double-clicking on the three dots), switch to the set of "Response" options, and change from "Try Again" to "Continue":

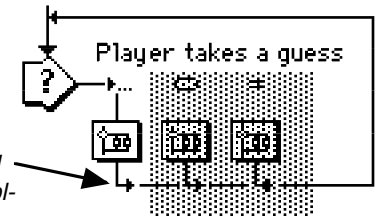


However, the second way is faster and better. It involves a little shortcut where you click on the flow line coming out of the branch while holding down the Command/Control key. This is the way we will do it.

Platform Alert!

As mentioned in chapter 1, Macintosh users use the Command key and Windows users use the Control key for all keyboard shortcuts. (No further reminders will be given in this chapter.)

☐ **While holding down the Command (Macintosh) / Control (Windows) key, click on the flow line just under the Map icon titled "*" until...**

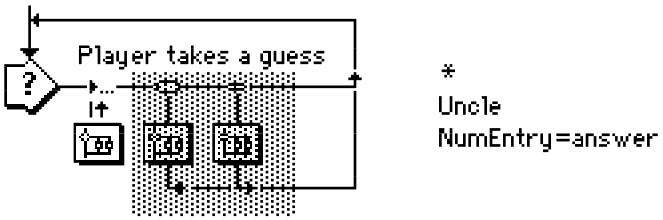


This is the point where you should Command/Control-click.

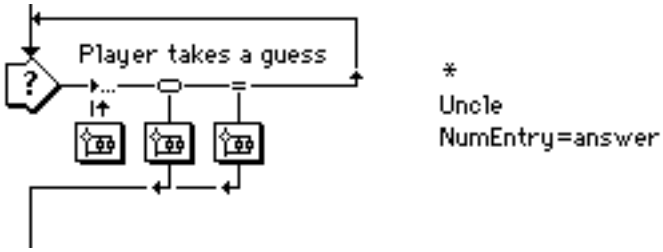
*
Uncle
NumEntry=answer

The branch paths just keep repeating if you continue Command/Control-clicking at this point.

...the flow line matches the following:



☐ **Command/Control-click on the other branches until the entire interaction looks like the following:**



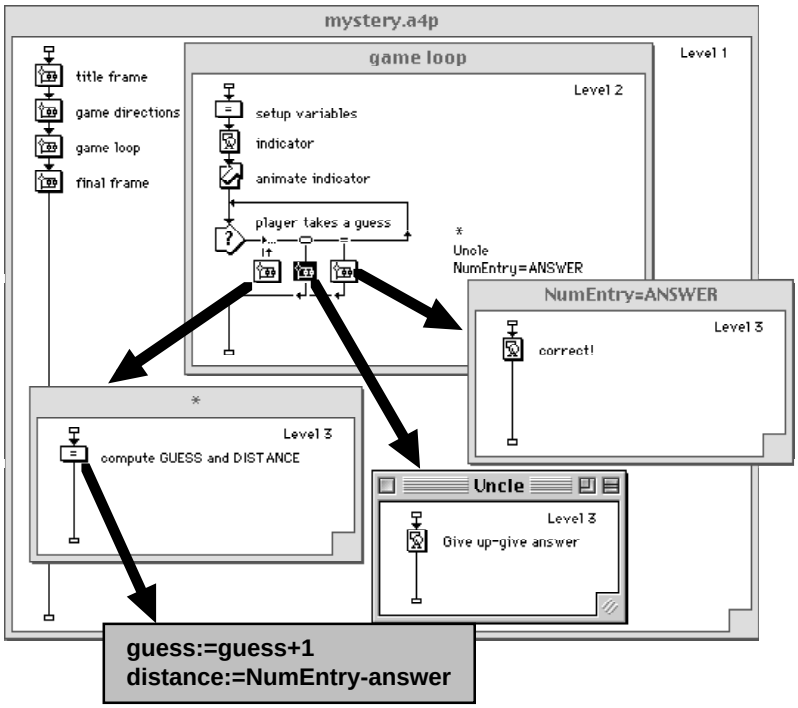
Double-check to be sure the branch paths in your interaction are exactly the same as these. The rest of the file depends on it.

Here is an "English translation" of what Authorware does when it encounters this Interaction icon and its three branches. It first displays whatever is inside the Interaction icon "Player takes a guess." Next, it waits for the player to either type in something and press RETURN or click the "Uncle" button. If the "Uncle" button is clicked, the flow is sucked into the Map icon "Uncle," executes whatever it finds inside, and then exits out of the interaction group. Let's say the player instead types something and presses RETURN. Since the title of the text response branch is "*," anything that the player typed will be a match. Authorware then stores the first number it finds in the player's answer in the system variable "NumEntry". The flow is then sucked into the Map icon "*" and the file executes whatever it finds inside. When finished, the flow exits the Map icon and continues back up to the original flow line. Authorware then makes its way to the equal sign of the third branch titled "NumEntry=answer." It checks if this condition is true. If it is, then the flow is sucked into this Map icon, executes whatever is inside, then exits out of the interaction group. If it is not a match, Authorware bypasses this branch and the flow continues on back around again to the Interaction icon "Player takes a guess" at which point the player is given another try.

Completing the game

Let's now turn our attention to the other tasks, such as having Authorware keep up with the number of guesses taken so far and the animation.

Here's a graphical overview of what's left to do:



Let's start by adding the contents to the Map icons under each of the three branches. Every time the player tries to guess the mystery number, we need Authorware to add one to the current value of the variable "guess." We also need to determine how close the guess is from the mystery number and store this information in another variable. The best place to do these tasks is inside the text response branch titled "*" (the first branch).

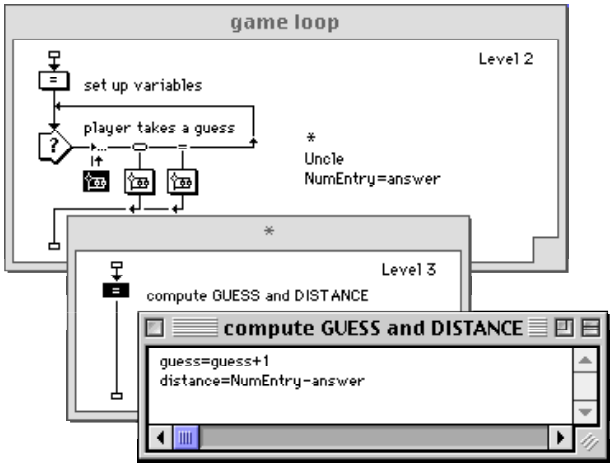
☐ **Double-click on the Map icon titled "*".**

This opens the Map icon and gives you a Level 3 flow line.

☐ **Drag a Calculation icon to this Level 3 flow line and title it "compute GUESS and DISTANCE."**

☐ **Double-click on this Calculation icon and type "guess=guess+1" on the first line in the window and "distance=NumEntry-answer" on the second line.**

Your screen should resemble the following:



The first line is an assignment statement that tells Authorware to "make GUESS what it was before plus 1."

Remember, these are assignment statements and not algebraic equations. Algebraically, guess=guess+1 is nonsense because you cannot have the same variable on both sides of an equation.

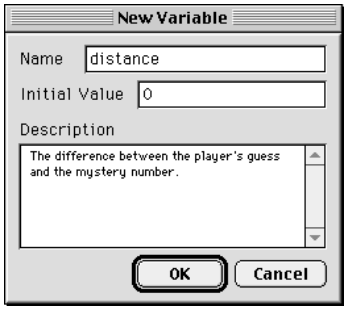
The second line is an assignment statement that creates the final variable used in this game — "distance." As you can see, Authorware subtracts the value of "answer" (the mystery number) from the variable "NumEntry" (the player's guess). The result will be a value to show "how far" the player's guess is from the correct answer. For example, let's say that the mystery number is 205. If the player guesses 300, then distance will equal +95. If the player guesses 100, then distance will equal -105. Obviously, all positive numbers will be above the mystery number and all negative numbers will be below the mystery number. This fact will figure prominently when we construct the animation in a few minutes.

☐ **When finished, close the "compute GUESS and DISTANCE" window.**

☐ **Click on the "Yes" button to save the changes to this icon.**

Since "distance" is a new variable, the "New Variable" dialog box automatically opens.

☐ **Give the variable “distance” an initial value of 0 and type a description similar to the following:**



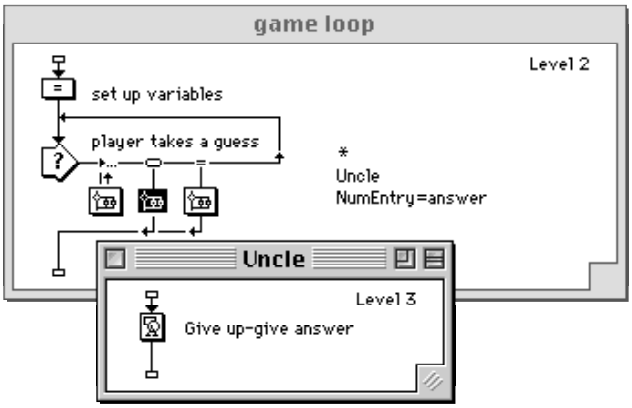
☐ **Click “OK”.**

☐ **Close the “*” Level 3 window.**

Now let’s work on the “Uncle” button branch.

☐ **Double-click on the Map icon titled “Uncle.”**

☐ **Drag a Display icon to this Level 3 flow line and title it “Give up-give answer.”**



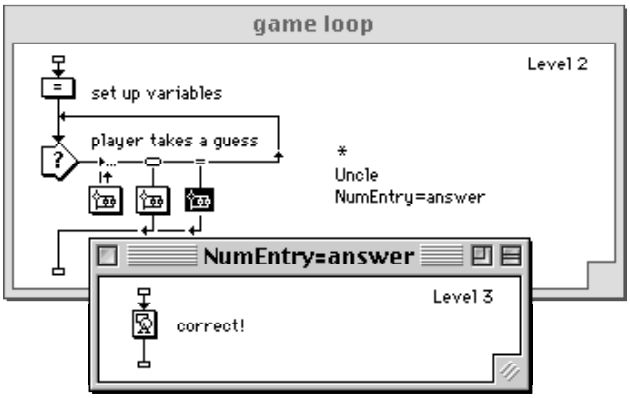
We could open this Display icon now (by double-clicking on it), but we are going to wait. As you’ll see, we can take advantage of the Authorware feature (introduced in previous chapters) where it automatically pauses the execution of the file when it encounters an undefined icon while it is being run. This will save us time.

☐ **Close the “Uncle” Level 3 window.**

Now let’s work on the third branch (conditional response type). This branch is triggered only when the player correctly guesses the mystery number. When this happens, we need to congratulate the player.

☐ **Double-click on the Map icon titled “NumEntry=ANSWER.”**

☐ **Drag a Display icon to this Level 3 flow line and title it “correct!”**



Again, we could open this Display icon now (by double-clicking on it). However, we are going to wait again as well.

☐ **Close the “NumEntry=answer” Level 3 window.**

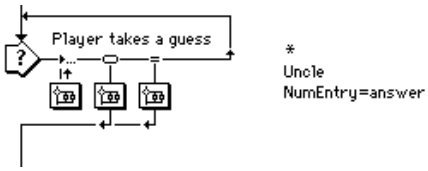
☐ **Save the file.**

Remember to try to get in the habit of saving on a regular basis. In the event of a power outage or system crash, you are always at risk of losing whatever you’ve done since your last save.

Performing a quick test

Before we construct the data-driven animation, it’s probably a good idea to test the file to make sure that the variables are working and programming logic is appropriate.

To do this, we need an easy way to check the current value of all the variables while the file is running. One trick is to temporarily display these variables on the screen.

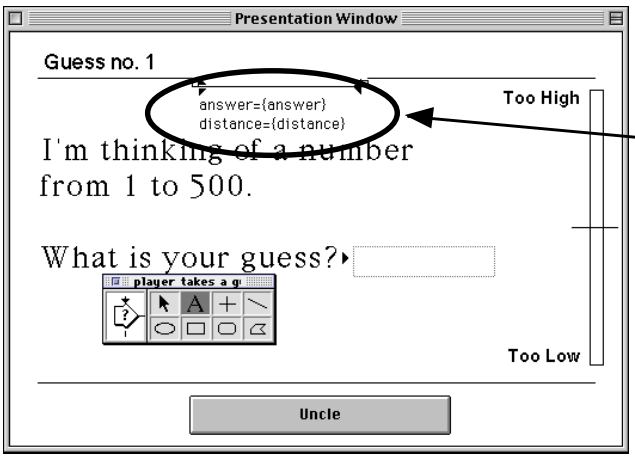


☐ **Double-click to open the Interaction icon “player takes a guess.”**

This shows the display of the Interaction icon.

☐ **Choose the text tool from the tool box.**

☐ **Click on a blank area of the screen and type the following:**



Add these lines. (Don't worry, we'll delete them later.)

☐ **Click on the pointer in the tool box.**

The current contents of these two important variables will be displayed (both should be 0).

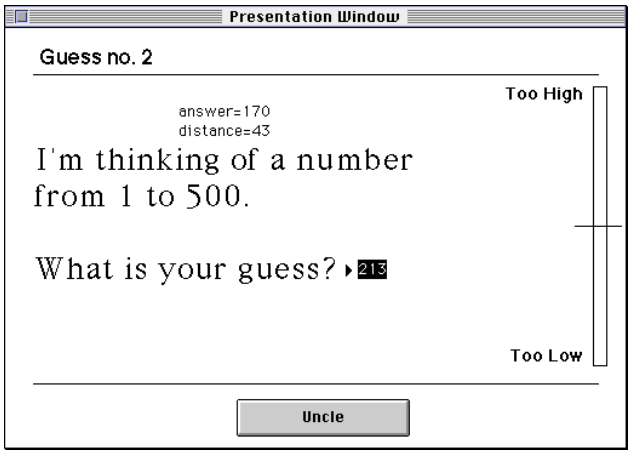
☐ **Click in the close box in the tool box to go back to the flow line.**

OK, let's give the file a try.

☐ **Select "Restart" from the "Control" menu.**

☐ **Type in a number *other than* the value of "answer" (the mystery number) and press RETURN.**

Look carefully at your screen:



Obviously, the value of "answer" will almost certainly be different on your screen since your computer chose its own random number from 1 to 500.

There should be a 2 beside "Guess no." Also, the variable "distance" should be exactly equal to the player's guess minus "answer."

What should you do if they are not working properly? Well, you are going to have to go back to find your error.

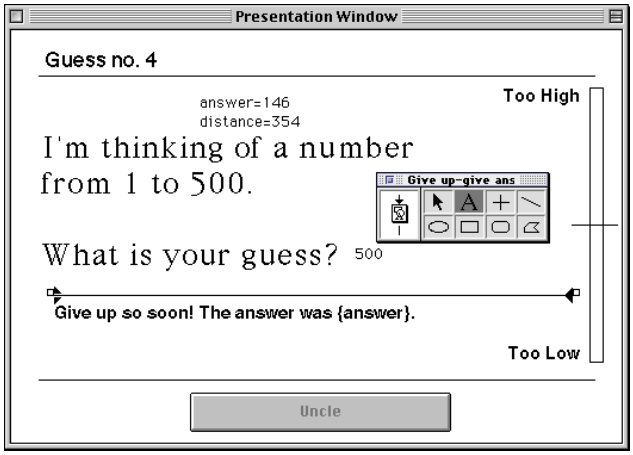
☐ **Continue to enter guesses *other than* the value of "answer" to make certain that the variables are functioning properly.**

While running the file, we can now easily and quickly enter the displays for the two yet undefined Display icons ("Give up-give answer" inside the "Uncle" Map icon and "correct!" inside the "NumEntry=ANSWER" Map icon).

☐ **Click on the Uncle button.**

Authorware will pause the execution of the file anytime it encounters an undefined icon — the Display icon titled "Give up-give answer" in this case. You will now see the tool box for this Display icon on your screen.

☐ **Click on the text tool and type the following on the screen:**



Change the font, size, style, and color to suit your own design.

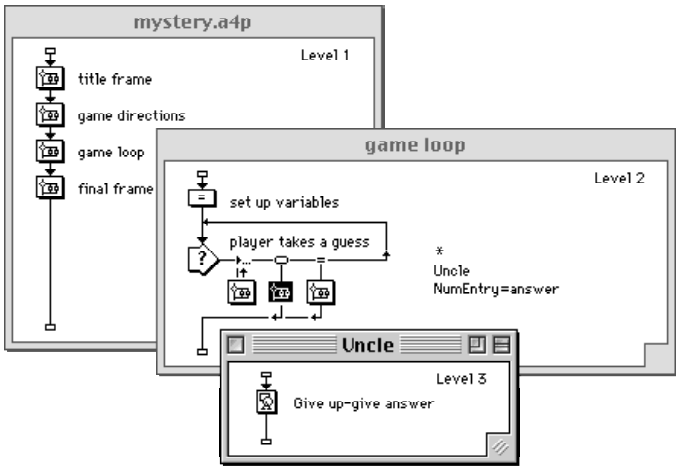
☐ **Click on the pointer in the tool box.**

The "mystery number" should appear in the line you just typed.

☐ **Click in the close box in the tool box to resume execution of the file.**

The screen will probably go blank. The reason can be found by studying the flow chart.

☐ **Press Command/Control-J to go back to the flow line.**



Follow the flow starting with the "baby pushbutton" right above the second branch titled "Uncle." When the player pushes the Uncle button, the flow is taken down this branch and into the Map icon "Uncle" and executes whatever is inside — the Display icon "Give up-give answer." Then, the flow continues to the bottom of the Level 3 flow line — this is equivalent of exiting the "Uncle" Map icon back on the Level 2 flow line. There the flow continues left and down to the bottom of the Level 2 flow line — this is equivalent to exiting the "game loop" Map icon back on the Level 1 flow line and into the "final frame" Map icon. Recall that we did not put any icons in there (double-click and look inside if you want). Therefore, the flow quickly goes in and out of the "final frame" Map icon and finally continues down to the bottom of the Level 1 flow line — this is equivalent to the end of the file.

Let's run the file again and this time we will get the right answer.

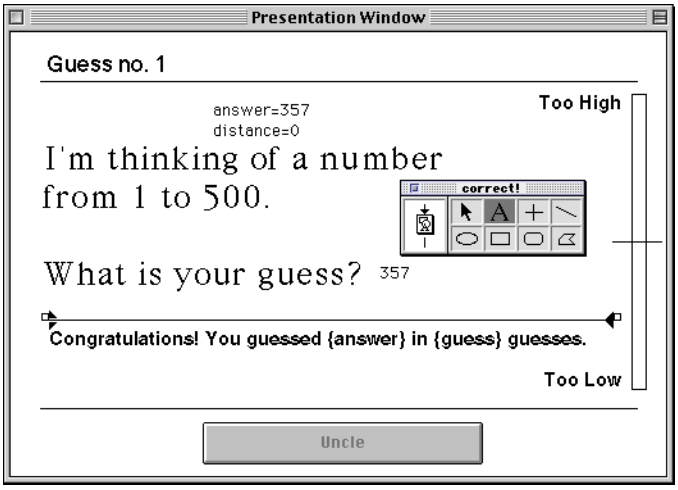
☐ **Select "Restart" from the "Control" menu.**

☐ **Enter a guess exactly equal to "answer" and press RETURN**

This will trigger the third branch involving the conditional response type. The Display icon titled "correct!" inside this branch should automatically open on the screen (look for the Display icon's tool box).

If it doesn't, go back to the flow line and double check the title for this branch.

☐ **Click on the text tool and type the following on the screen:**



☐ **Click on the pointer in the tool box.**

The sentences you just typed should read "Congratulations! You guessed 357 in 2 guesses."

Obviously, it is unlikely that your screen will read "357" since this number is randomly chosen by the computer.

Why does it say we guessed it in 2 guesses when the number of guesses shown on the top of the screen still shows 1? Well, recall that we add one to guess immediately after the player presses RETURN. Since the initial value of "guess" is 1, it *should* be 2 at this point. Obviously, we'll have to fix this in a moment (by subtracting one from "guess" when the player gets the right answer. The guess counter at the top of the screen still reads "1" because the flow never returned to the Interaction icon "Player takes a guess." Therefore, the display was never updated and still shows the old value.

☐ **Click in the close box in the tool box to resume execution of the file.**

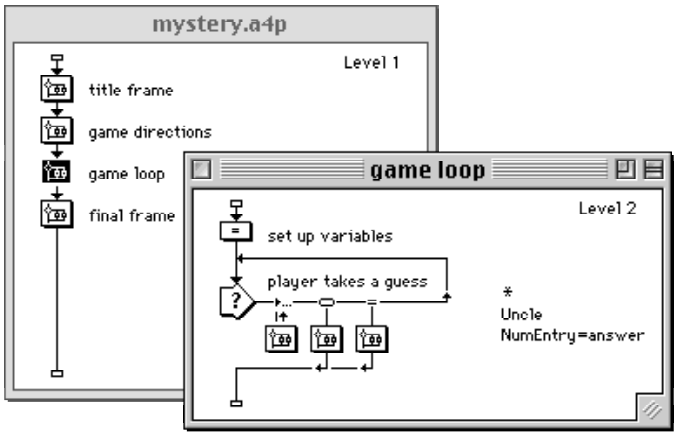
The screen will again go blank for the same reasons previously discussed.

☐ **Restart the file at least two more times: once to make sure the "Uncle" button works properly and once to make sure the correct answer condition works properly.**

You've probably noticed a second problem by now: the two displays are erased before the player has a chance to read them. Let's take care of this problem plus the problem associated with the variable "guess."

Debugging the file

☐ **Press Command/Control-J to jump back to the flow line.**

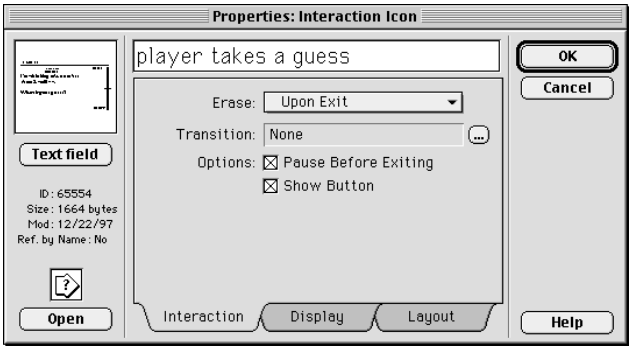


We need to get to a dialog box with special options for the Interaction icon "player takes a guess". As usual, there is a long way and a shortcut to get to this dialog box. The long way is to click once on this icon, then select "Icon" from the "Modify" menu, then select "Properties...". We'll show the shortcut next; it involves Command/Control-double-clicking on the icon.

☐ **Hold down the Command/Control key and double-click on the Interaction icon titled "player takes a guess."**

☐ **Click in the select boxes beside "Pause Before Exiting" and "Show Button"** (be sure you are in the "Interaction" set of options).

An "X" should appear in each box indicating that these options are turned on:



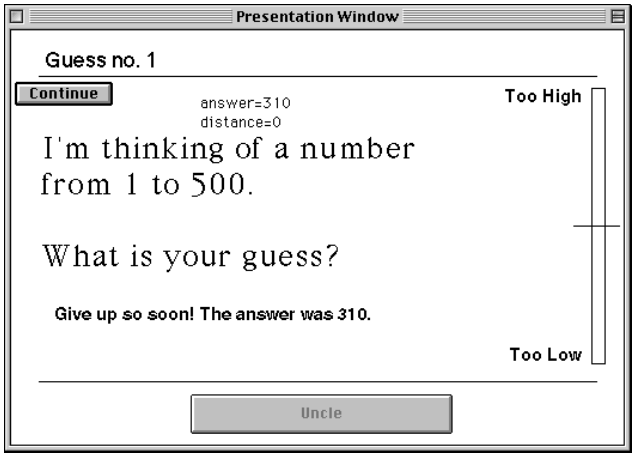
While here, notice that Authorware is set to "Erase Interaction: Upon Exit." That's why the screen goes blank at the end — this features erases all displays that happen to be on the screen produced during the interaction.

We are done — just click on "OK."

☐ **Click "OK."**

☐ **Run the file several times to make sure that the file pauses at the appropriate time to give the player time to read the displays.**

The file should pause either when the correct number is entered or the Uncle button is pressed.



You'll probably want to reposition the "continue" button. Here's how:

☐ **When the "continue" button appears on the screen, press Command/Control-P to pause the file.**

☐ **Drag the "continue" button to a screen location of your choice.**

☐ **Press Command/Control-P again to proceed.**

Now, let's fix the "guess" variable.

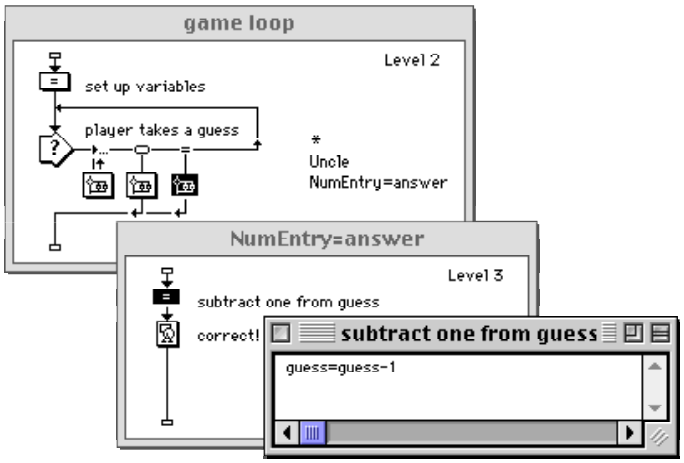
☐ **Press Command/Control-J to go back to the flow line.**

☐ **Double-click on the Map icon "NumEntry=answer."**

☐ **Drag a Calculation icon to the Level 3 flow line just above the Display icon titled "correct!" and title it "subtract one from guess".**

An even better design would be to make the "Continue" appear in place of the "Uncle" button. You could just enlarge the continue button to the same size and place it over the Uncle button. But it would be better to tell Authorware to "Hide" the "Uncle" button when it is inactive. To do this, just open up the options dialog box for the "Uncle" branch (by double-clicking on the baby pushbutton on the flow line).

☐ **Double-click on this Calculation icon and type the following in the text window:**



Be sure to put this Calculation icon above (i.e. before) the Display icon "correct!"

☐ **When finished, click on the close box in the Calculation icon's window.**

☐ **Click on the "Yes" button to save the changes to this icon.**

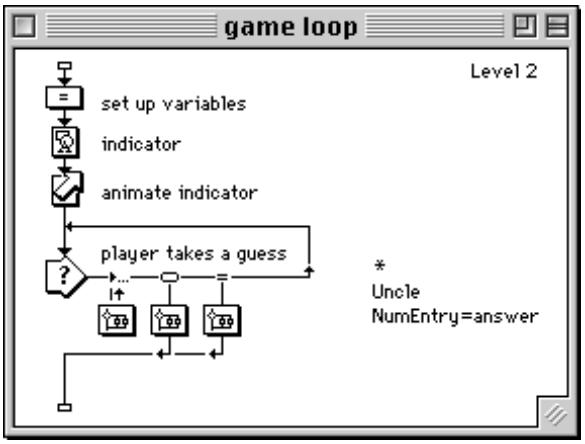
This should solve the problem.

☐ **Run the file several times to make sure everything is now functioning properly.**

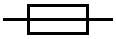
Constructing data-driven animation as feedback

The next thing to do is the animation. This will not take long since the hard part is already done — setting up the variable "distance." This variable will provide all the data we need to control the animation.

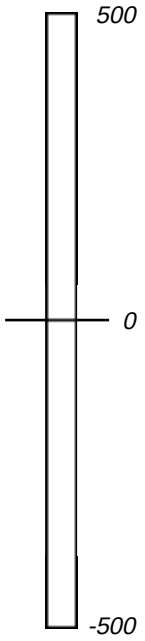
☐ **Drag a Display icon and a Motion icon to the spot on the Level 2 flow line indicated below and title them "indicator" and "animate indicator" respectively:**



The "indicator" will consist of a very simple display, something like the following:



Feedback scale



This display will slide up and down on the feedback scale a certain distance away from the center line depending on the player's guess. If the guess is too high, this indicator should be above the center line. If the guess is too low, it should be below the center line. Fortunately, we already have a variable, "distance," that contains this information (assuming that the feedback scale is marked off according to the graphic at the left).

Remember that the variable "distance" contains a value equal to the difference between the player's guess ("NumEntry") and the mystery number ("answer').

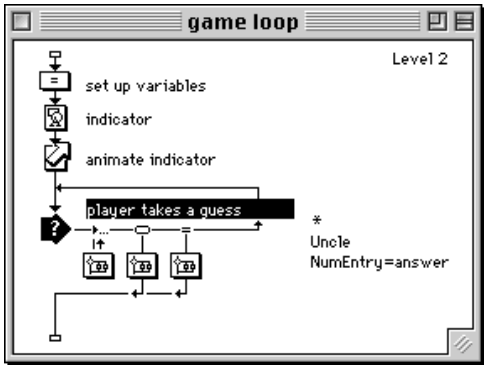
The tricky part is getting the indicator perfectly aligned with the feedback scale. Therefore, it would be most helpful if the feedback scale were displayed while we define this Display icon and the animation icon that follows. The first thought is to just run the file and let Authorware automatically open both the "indicator" and "animate indicator" icons. The problem is that the feedback scale is only displayed in the Interaction icon titled "player takes a guess" which comes *after* these two icons.

There are a couple of strategies. One is to create another Display icon, called something like "feedback scale," which is placed just before the "indicator" icon. The existing feedback scale graphic could just be "cut and pasted" into it.

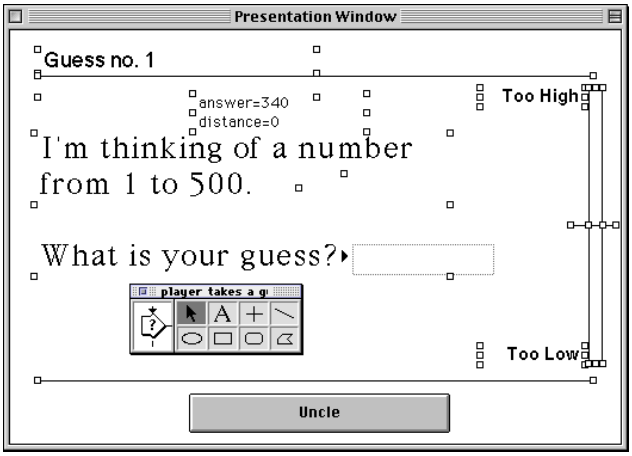
Instead, we'll leave the displays the way they are and use this little problem to learn another nice Authorware trick for showing other displays while working on a new display. It involves getting Authorware to "remember" the last thing displayed and then "shift-double-clicking" on *another* display (or animation) icon. Let's do it.

The first step is to get Authorware to display the contents of the Interaction icon titled “player takes a guess.”

☐ **Double-click on the Interaction icon “player takes a guess.**



The contents of the Interaction icon are displayed on the screen (including the all important feedback scale):



A special feature of Authorware is that it always stores the very last thing displayed on the screen in memory.

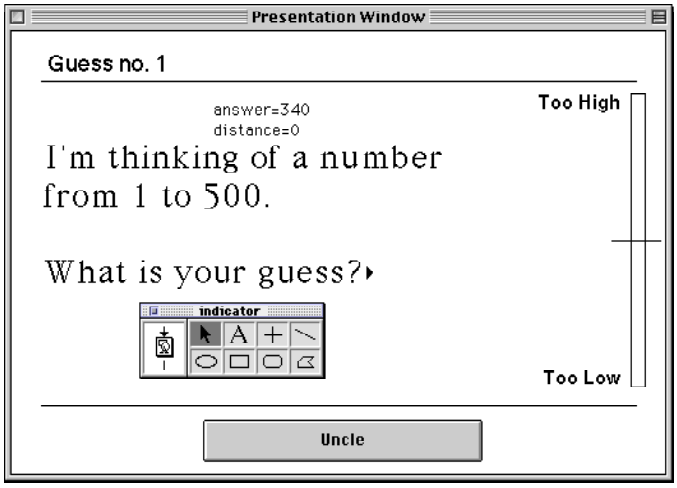
☐ **Click in the close box in the tool box (or just press Command/Control-J) to go back to the flow line.**

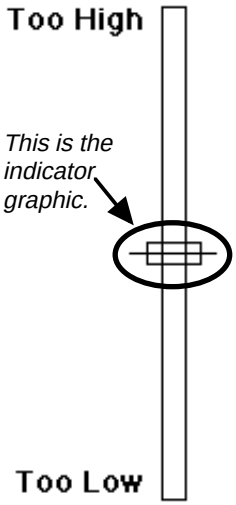
Although the presentation window is gone, Authorware continues to “remember” what was just displayed.

☐ **Hold down the shift key and double-click on the Display icon titled “indicator.”**

This tells Authorware to open the Display icon “indicator” *plus* show whatever was most recently displayed on the screen.

Although the screen looks identical to that displayed just seconds ago, notice that the tool box's title shows "indicator" — the title of the new Display icon. Any graphic or text object we place on the screen now will be contained within the Display icon "indicator."





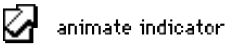
☐ **Construct the indicator graphic with the rectangle and straight line tools and place it precisely at the midpoint on the feedback scale.**

Precision counts here!

Remember that you can use the arrow keys on the keyboard to fine tune the position of any selected graphic one pixel at a time.

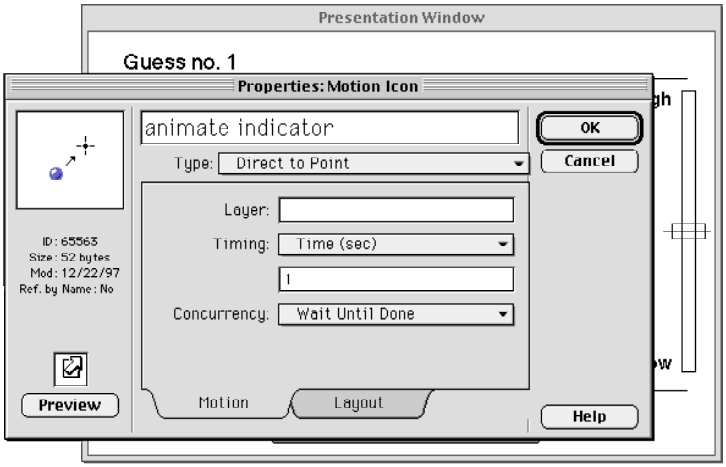
☐ **When finished, click in the close box in the tool box to go back to the flow line.**

Next we need to construct the animation. We want both the indicator and feedback scale displayed on the screen as we define the Motion icon. Fortunately, those two displays were the last thing shown on the screen, so they are now both in Authorware's memory.



☐ **Double-click on the Motion icon titled "animate indicator."**

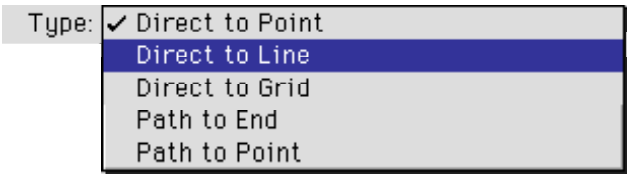
The Motion icon's dialog box should appear along with both the indicator and feedback scale displays:



We can now proceed to construct the animation for the indicator while using the feedback scale display as a guide.

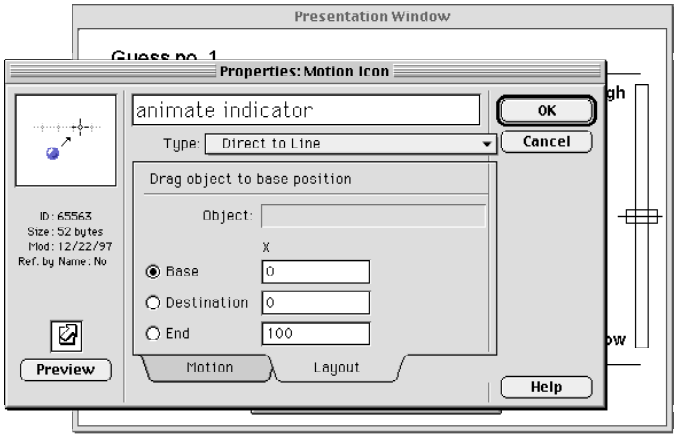
First, we need to change the motion type to one of the data-driven animation types.

☐ **Click and hold on the words "Direct to Point" to activate the pop-up menu, then select "Direct to Line":**

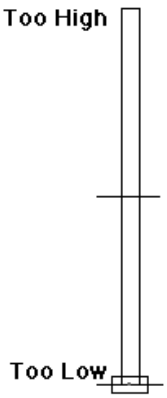


☐ **Click on the Layout tab to go to the Layout options.**

The motion's dialog box now changes accordingly:



The next step is to follow the directions in the dialog box:



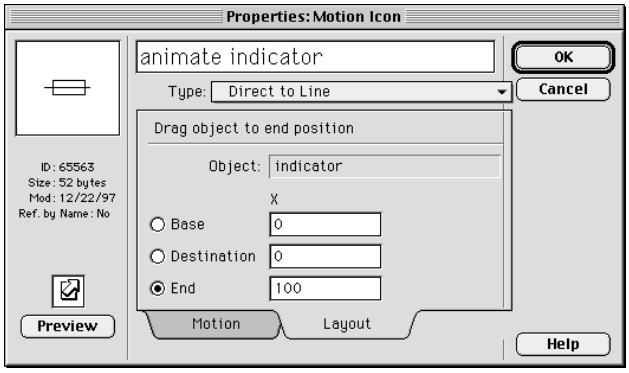
☐ **Drag the indicator to the "base position," that is, precisely to the bottom of the feedback scale.**

If you find it difficult to "grab" the indicator, remember that you must be clicking on one of the indicator's edges.

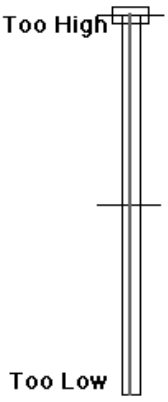
Now we need to define the end position.

☐ **Click in the radio button beside "End":**

Line up the indicator's center line with the bottom edge of the feedback scale.



Notice that the directions in the dialog box change accordingly.



☐ **Drag the indicator to the "end position," that is, precisely to the top of the feedback scale.**

Line up the indicator's center line with the top edge of the feedback scale.

Remember, you can click on the radio buttons beside "Base" and "End" to fine tune the positions. However, if you really messed up during these procedures, the best advice is to click "Cancel" and try again.

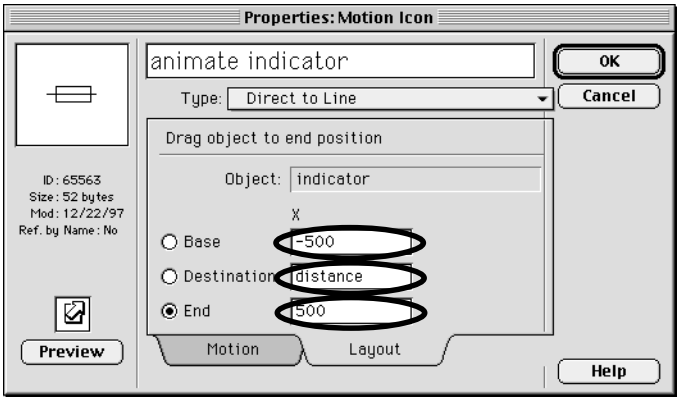
There are three more things we need to do: 1) tell the Motion icon that we are using the variable "distance"; 2) change the base and end values to 500 and -500, respectively; and 3) change the Motion icon to animate this indicator perpetually.

☐ **Type "distance" inside the text window beside "Destination."**

☐ **Type "-500" in the text window beside "Base."**

☐ **Type "500" in the text window beside "End."**

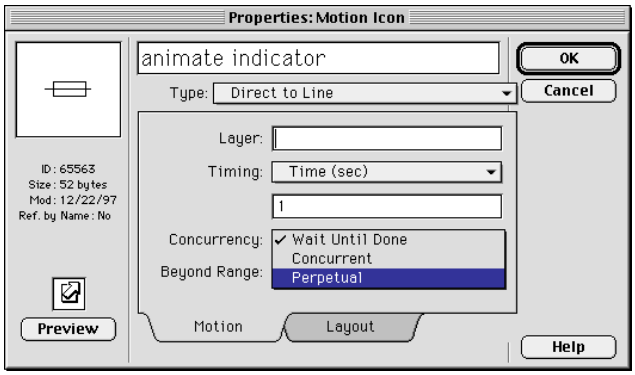
Your dialog box should look like the following:



☐ **Click on the "Motion" tab to display the Motion options.**

☐ **Click and hold on "Wait Until Done" to reveal the pop-up menu.**

☐ **Select "Perpetual" from the choices given:**



A perpetual animation means just that — the indicator will be animated *all the time.* This is a powerful feature of Authorware. As the variable "distance" changes, Authorware will automatically move the indicator to the new location without any additional programming on our part.

We're done!

☐ **Click "OK."**

This should send you back to the flow line. Time to test our game.

Testing the game

☐ **Restart your game several times to test the animation of the indicator.**

Enter guesses above and below the mystery number to make sure that the indicator animates to appropriate positions.

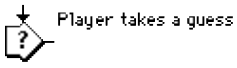
You'll notice that the indicator has a margin of error of about 3 pixels, depending on how long you drew the scale. If you try guesses that are only 1 or 2 numbers off, the indicator will probably look as though it is perfectly aligned with the scale. There are several ways to correct this, the simplest being to change the game so that the mystery number is somewhere between 1 and 100. Another strategy is to do more programming in order to recalibrate the "base" and "end" positions according to another set of variables.

We will leave the game as it is.

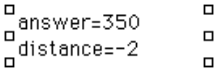
Mopping up

Obviously, we don't want to give the game to the fifth graders with the mystery number right there on the screen. Let's go back and delete the variables we temporarily displayed on the screen to help us test the game.

☐ **Press Command/Control-J to go back to the flow line (if you are not already there).**



☐ **Double-click on the Interaction icon "Player takes a guess."**



☐ **Click once on the text object showing the value of the variables "answer" and "distance."**

☐ **Press the "Delete" button on the keyboard.**

☐ **Click in the close box in the tool box to go back to the flow line.**

Let's save.

☐ **Select "Save" under "File."**

Adding a final frame

As is, the game works fine, but it ends very rudely —just a blank screen!

Let's go ahead and add a final frame that gives the player the chance to play again or quit. This involves setting up a simple interaction in the Map icon titled "final frame." We will also take advantage of two Authorware functions to either restart the game or quit the game and send the player back to the computer's desktop.

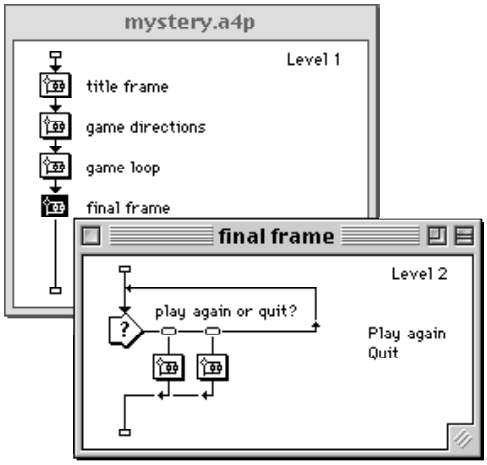
It is now assumed that you are comfortable with setting up Interaction icons with branches. The instructions that follow are therefore abbreviated.

☐ **Close all windows except the Level 1 flow line:**

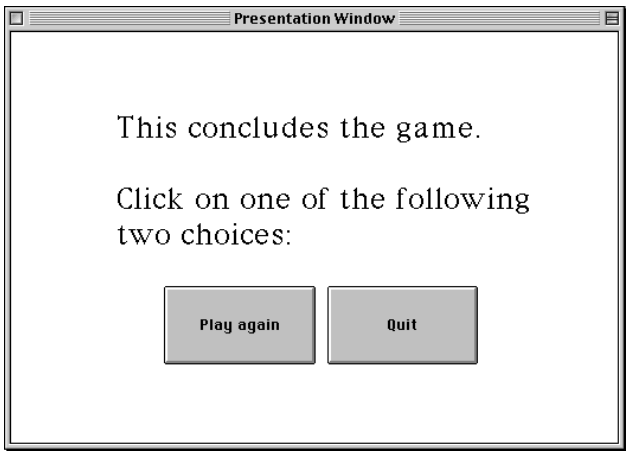
☐ **Double-click the Map icon "final frame."**

☐ **Drag an Interaction icon to the Level 2 flow line and title it "Play again or quit?".**

☐ **Continue constructing the interaction so as to have two pushbutton branches, as per the following:**



☐ **Double-click on the Interaction icon "play again or quit?" and construct the display to match the following** (jump back to the icons when done)**:**

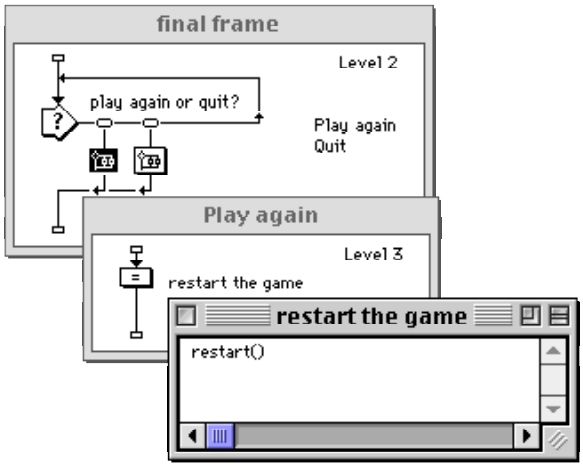


All we need to do now is embed the appropriate function into each of the two branches: Restart and Quit.

☐ **Open the "Play again" Map icon, drag a Calculation icon to the Level 3 flow line, and title it "Restart the game."**

☐ **Type the function "Restart()" into this Calculation icon.**

To learn more about this function (or to have Authorware paste it in for you), go to "Functions" under the "Window" menu. It is a general category function.



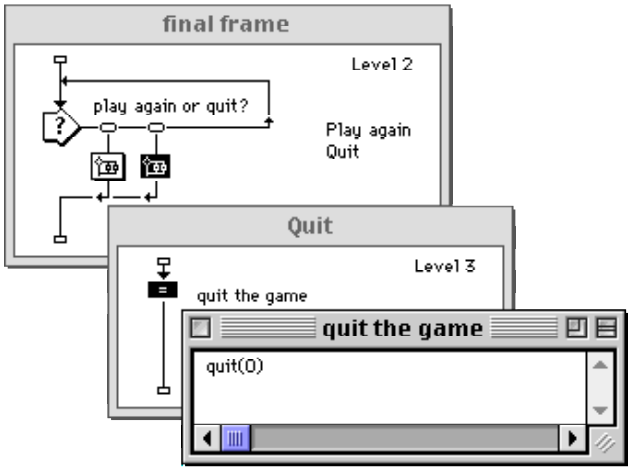
The Restart() function causes Authorware to branch to the beginning of the file. All variables are reinitialized (set back to their initial values).

☐ **When finished, click in the close box in the Calculation icon's window; click on the "Yes" button to save the changes to this icon.**

☐ **Open the "Quit" Map icon, drag a Calculation icon to the Level 3 flow line, and title it "Quit the game."**

☐ **Type the function "quit (0)" into this Calculation icon.**

To learn more about this function (or to have Authorware paste it in for you), go to "Functions" under the "Window" menu It is a general category function.



When Authorware encounters the Quit function in an unpackaged file, it stops running the file and displays the flow line. In a packaged file, Authorware will quit the file and then do one of several things, depending on the option defined by the number inside the parentheses. Here are descriptions of the options taken directly from Authorware:

> 0 = (Default) Quits Authorware and displays the Program Manager (Windows 3.1), Desktop (Windows 95 and NT 4.0), or Finder (Macintosh). If the Authorware file that quits was called by a jump from another Authorware file, Authorware now returns to the other file.
>
> 1 = Quits Authorware and displays the Program Manager (Windows 3.1), Desktop (Windows 95 and NT 4.0), or Finder (Macintosh).
>
> 2 = Restarts Windows (Windows 95 and Windows NT), exits to DOS (Windows 3.1), or restarts the computer (Macintosh).
>
> 3 = Shuts down Windows (Windows 95 and Windows NT), quits Authorware and displays the Program Manager (Windows 3.1), or shuts down the computer (Macintosh).
>
> Use Quit only in calculation icons.
>
> Note: When the Shockwave Plug-In is running a piece in nontrusting mode, it disables Quit(2), Quit(3), QuitRestart(2), and QuitRestart(3).

See Appendix E for information about Shockwave.

☐ **When finished, click in the close box in the Calculation icon's window; click on the "Yes" button to save the changes to this icon.**

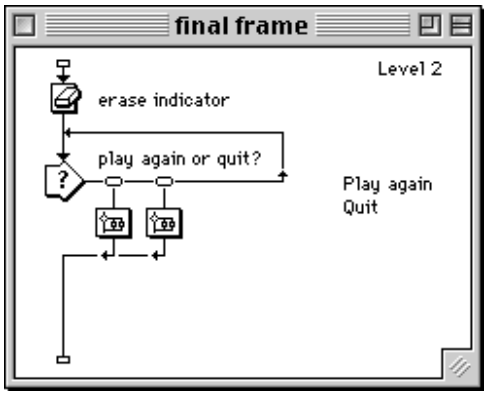
☐ **Restart the file several times to test these two functions.**

Now you'll have to play the game "for real"! See how many guesses it takes you.

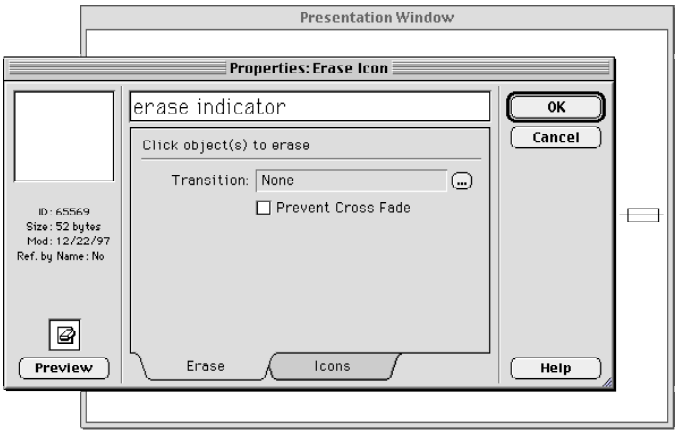
Fixing one final bug

You've probably noticed just one more little glitch — the indicator graphic does not erase when the game ends. This is because the Display icon "indicator" is not part of the interaction group, therefore it is "immune" to the automatic erasing. We will have to erase this icon manually.

☐ **Drag an Erase icon to the top of the Level 2 flow line of the "final frame" Map icon and title it "erase indicator."**



☐ **Restart the file to the end of the game, click on the indicator graphic when the Erase icon's dialog box appears, then click "OK."**



☐ **Restart the file several more times to make sure everything works properly.**

☐ **Save the file.**

☐ **Package the file.**

See Appendix A for background and instructions on packaging.

☐ **Go have some fun with the fifth graders!**

Summary

In this chapter you created a simple math game that randomly selects a "mystery number" from 1 to 500. Players can take as many guesses as they want to determine the mystery number or they can click on an "Uncle" button to give up and try again. You learned more about variables and functions, including how to create data-driven animation to provide feedback to the player. You learned how to embed and display variables on the screen. You also learned how to create an interaction that relies on a conditional response type. Finally, you learned much about game design.

Other projects

1. Go back and change or enhance the graphics, especially the feedback scale and indicator.

2. Change the range of possible mystery numbers, such as from 1 to 100, 10 to 20, or 0.00 to 1.00.

3. Give players the choice of determining the upper range of mystery numbers (from 1 to ?).

4. Include sound as additional feedback: one sound when the guess is too high, one when it is too low, and a special sound when the player guesses correctly.

5. Change the feedback scale to a single "hot and cold" version without any information about the guess being too high or too low. (Hint: You will have to use the absolute value of the variable "distance.")

6. Invent a new and creative context for the game. For example, instead of trying to determine a "mystery number," have the player pretend they are looking for a lost mountain hiker — the goal is to rescue the hiker in as few days (i.e. guesses) as possible with their "radar" (i.e. feedback scale).

Chapter 4: Space Shuttle Commander

Building a physics simulation

This chapter shows how to create a simple physics simulation that uses data-driven animation.

In this session you will be saving the following files:

SSC1.a4p
SSC2.a4p
SSC3.a4p

This chapter assumes you are familiar with the structure of Authorware files, the use of the flow line, and the Map, Display, Erase, Interaction, Calculation, and Motion icons. It also assumes that you have an introductory knowledge of how to create and use variables and system functions, similar to what was presented in Chapters 2 and 3.

The Problem

You are a middle school science teacher. Your class has been studying Newton's laws of motion. The textbook keeps talking about the motion of objects in "ideal" situations, such as when objects are no longer under the influence of friction or gravity. You notice that some of the students seem really interested in some of the textbook's examples, such as the motion of the planets and astronauts walking in space. But it's become pretty clear that their understanding of these laws is rather shallow, but you know it's not their fault since they have never known a world without friction or gravity. You want your students to *experience* the laws of motion, not just study them. Maybe Authorware can help.

I almost hesitate to reveal how to build this project — after all, I've built a hearty career doing research on versions not much more sophisticated than the one you're about to construct.

Introduction

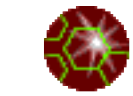
In the previous two chapters, you learned how to create variables to control the animation of screen objects. The previous chapter used data-driven animation as feedback in a math game. The project in this chapter also depends on data-driven animation as feedback under the control of the user. In addition, this project combines both gaming and simulation techniques. Although the content may sound intimidating — building a physics simulation — the mathematical model of this simulation again uses surprisingly simple arithmetic to build a working prototype. However, we will also go a little bit further in this chapter to show how some slightly more advanced mathematics can greatly enhance the simulation.

We will be saving three different versions of the simulation as we go. The context of all three versions is the space shuttle. Our goal will be to elicit the fantasy of the being the space shuttle's commander. The first version, SSC1.a4p, will be a working prototype of the simulation, but without any gaming features. SSC2.a4p will expand this prototype to include a simple game in which the player tries to "rendezvous" their space shuttle with a space station. SSC3.a4p will improve the game by using a mathematical approach to determine if the shuttle has successfully docked with the space station.

Although this chapter will again guide you step-by-step through the construction process, we will begin to assume you are beyond the introductory level. Instructions will be a bit more abbreviated except when a new procedure is introduced.

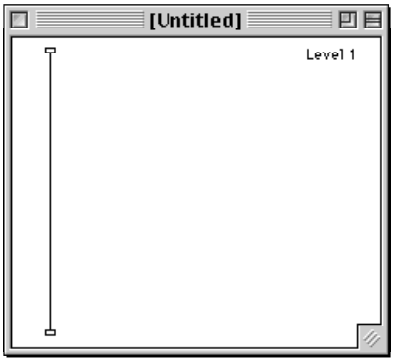
Getting started

The first step is to launch the Authorware application.



☐ **Start Authorware by double-clicking on the Authorware application.**

A new file opens with a Level 1 flowchart with the name "untitled":

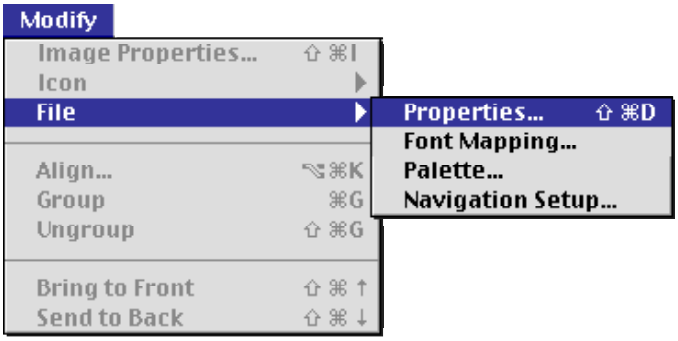


Changing the background color

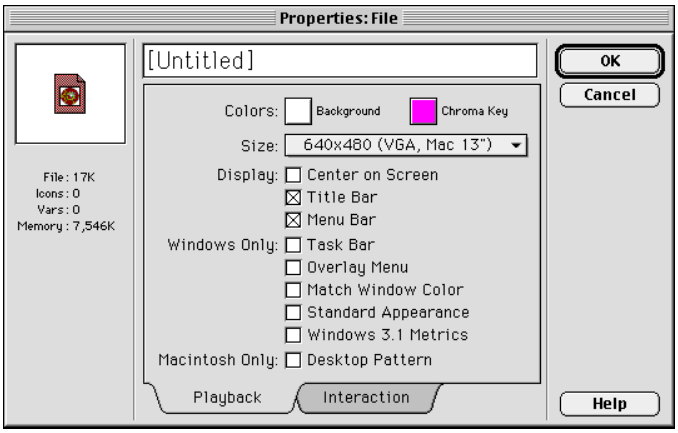
It will be important throughout this chapter to consider ways to trigger and maintain the fantasy of being the commander of the space shuttle. A good start might be to change the background color of the screen from the default color of white to something more "spacey." Although black might be most appropriate, let's choose a dark blue instead.

You can only have one "true" background color for the entire file. However, you can get around this limitation simply by using a Display icon as to show a background graphic, even something as simple as colored rectangle that fills the entire screen. As long as this Display icon comes before other icons and is not erased, the screen's background will stay that color for the entire file (or until you choose to erase and replace it with another).

☐ **Select "File" under "Modify" from the menubar, then select "Properties...".**



This opens a dialog box with options pertaining to the entire file (be sure you are viewing the "Playback" options):

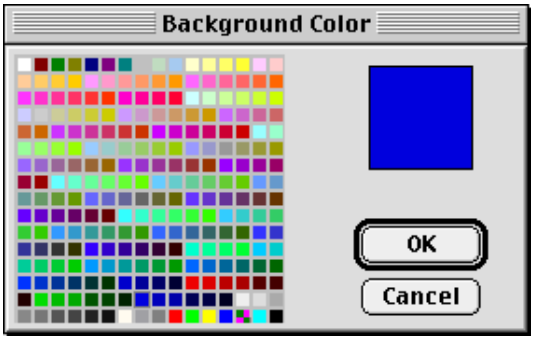


All we want to change is the background color.

☐ **Click on the white square between "Colors:" and "Background".**

A color palette appears. The number of color choices depends on how many colors your monitor is set to. Choose any dark blue of your choice.

☐ **Click on a shade of dark blue, then click “OK.”**



Sorry, no color printer! You’ll have to pretend this is dark blue.

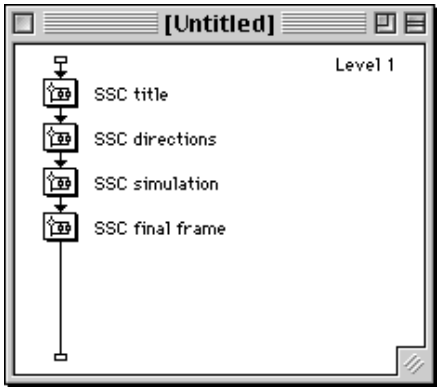
This sends you back to the File Properties dialog box in which background color you just chose now appears.

☐ **Click “OK.”**

Setting up the game structure

Just like last chapter, we will start is by using Map icons to organize the overall structure of the simulation parts.

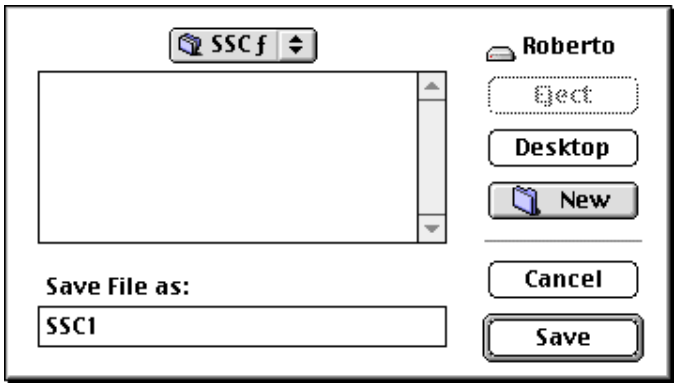
☐ **Drag four Map icons to the flow line and title them as per the following:**



Let’s go ahead and save the file.

☐ **Select ”Save as...” from the File menu.**

☐ **Save your file with the name "SSC1" to the folder of your choice.**

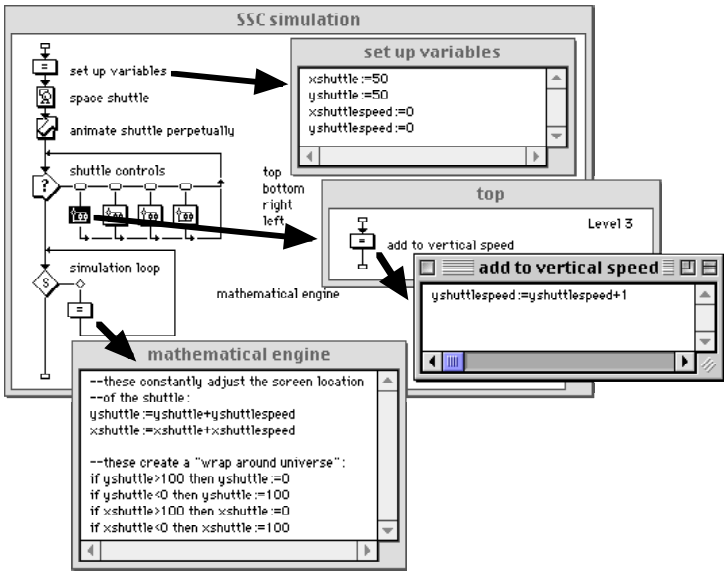


From here on, remember to save your file periodically using the "Save" command from the "File" menu.

Note again that Authorware will automatically add the extension ".a4p" to your file name.

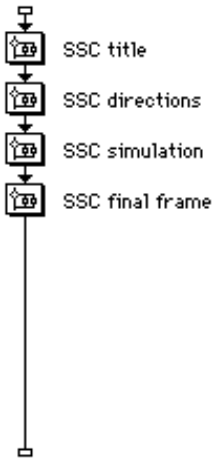
Like last chapter, we are not going to take the time to step through the design of a suitable title and directions. Instead, we want to again dive right into the construction of the simulation itself.

Here is a graphical overview of the finished simulation:



This should help orient you to where we are going with this. Note, however, that only the first button branch off of the "shuttle controls" interaction is shown. However, the logic is similar in the other branches.

Let's get started.



☐ Double-click on the Map icon "SSC simulation."

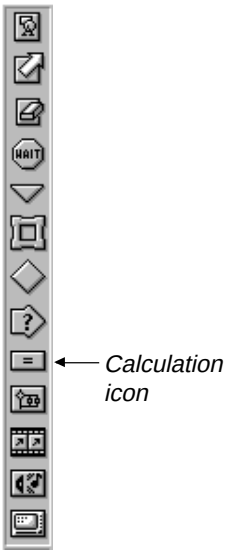
The first thing we will do is set up all the variables that this simulation will use. Amazingly, only four variables are needed to build a working prototype. Here's a description of the variables that are needed:

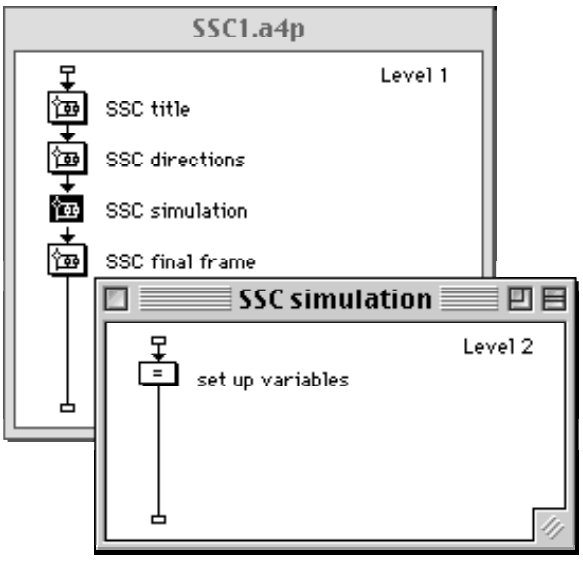
XSHUTTLE- contains the current horizontal position of the space shuttle.
YSHUTTLE- contains the current vertical position of the space shuttle.
XSHUTTLESPEED- contains the current horizontal speed of the shuttle. For example, if it is 0, then the shuttle is not moving at all side to side. If it is 1, then the shuttle is slowing moving from left to right. If it is 2, then it is moving from left to right, but faster. If it is a negative number, such as -3, then the shuttle is moving from right to left.
YSHUTTLESPEED- contains the current vertical speed of the shuttle. Positive numbers mean that the shuttle is moving from bottom to top and negative numbers means that the shuttle is moving from top to bottom.

All of these are numeric variables. Although it's easy to get confused at first, these variables simply define the three basic properties of moving objects: distance traveled, velocity (speed and direction of the object, in other words, change in distance), and acceleration (change in velocity). "xshuttle" and "yshuttle" define the distance traveled, and "xshuttlespeed" and "yshuttlespeed" define the shuttle' velocity. Changes to velocity (by way of the player's clicking) define acceleration.

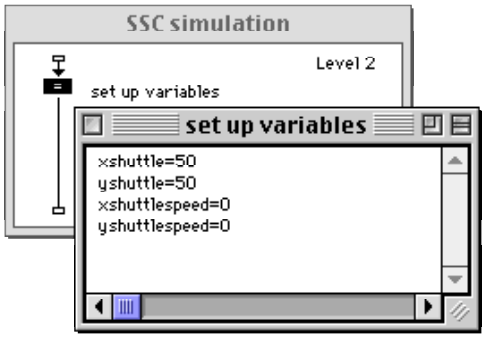
Understanding the relationship between distance, velocity, and acceleration is a wonderful introduction to physics and advanced mathematics. Historically, these sorts of motion problems were the motivation behind many of the earliest scientific discoveries by Galileo, Kepler, and Newton. In fact, Isaac Newton invented the calculus as a type of mathematical "short cut" for solving these problems more quickly and easily.

☐ Drag a Calculation icon to the Level 2 flow line and title it "set up variables."





☐ **Open this Calculation icon and type the following in the text window:**

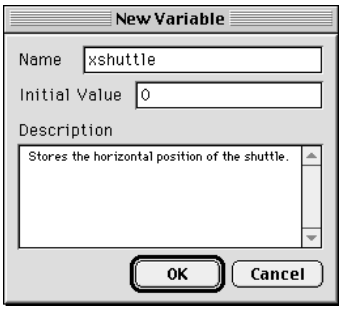


The first two lines will give the shuttle an initial position of 50 across and 50 up on a two-dimensional grid going from 0 to 100 that we will soon be constructing for the shuttle's animation. The last two lines give the shuttle an initial horizontal and vertical velocity of 0. In other words, we want the shuttle to be at the center of the screen's "universe" and not moving at the start of the simulation.

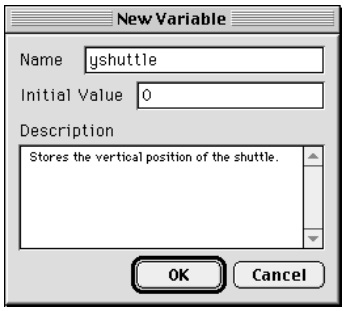
☐ **Click on the close box to close the Calculation icon's window, then click "yes" to save the calculation changes.**

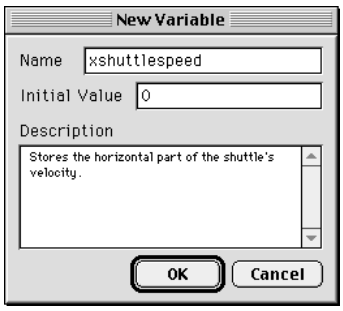
You will need to define each of these variables for Authorware because this is the first time it has encountered them.

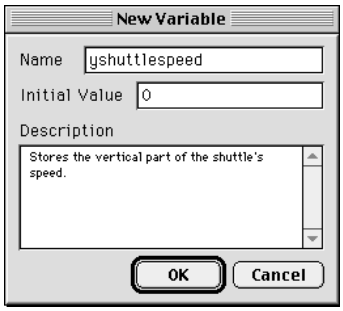
☐ **Type in the following initial values and descriptions for each of the following four new variables, and click "OK" after each one:**



It could be argued that we should enter 50 for these first two variables since that is the value they are set to above. However, 50 is just a good starting point which may change in later versions.



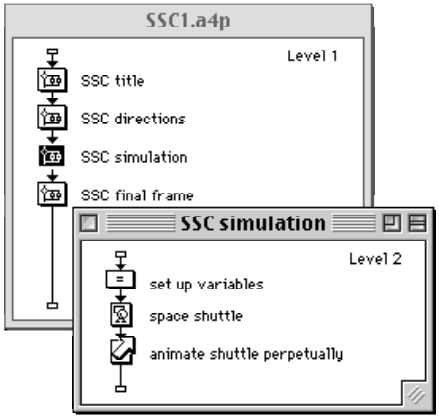




Constructing the shuttle display and its animation

Next, let's construct the shuttle display and the Motion icon that will control its animation.

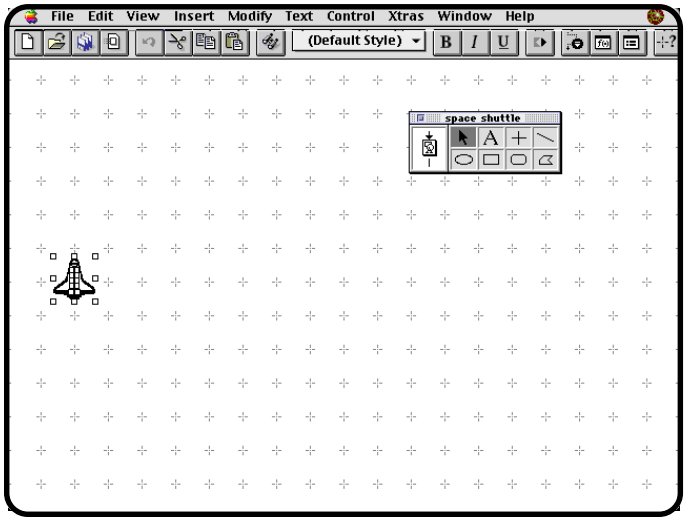
☐ **Drag a Display icon and a Motion icon to the Level 2 flow line and title them as per the following:**



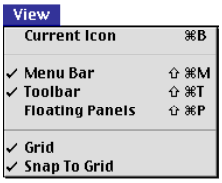
☐ **Double-click on the Display icon "space shuttle."**

☐ **Construct a suitable "space shuttle" graphic and put it roughly at the left edge of the screen.**

As warned in previous chapters, don't get bogged down at this point trying to get the world's most perfect graphic. If you don't have any graphic ready, just put a yellow ball or block on the screen as a "graphic placeholder." You can always go back later and replace this graphic without doing any harm to the animation. A copy of the shuttle graphic shown to the left is in the Rieber Clip Media resource.



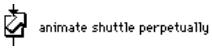
Choose to view the grid and turn on the "Snap to Grid" option as needed:



You can copy and paste this graphic from the "clip art" file accompanying this book, or you might construct your own space shuttle graphic with a graphics application, then copy and paste it into this Authorware file.

In fact, a great strategy would be to have the middle school students draw their own space shuttles and starships to be "cut and pasted" into the simulation.

☐ **When finished, click on the close box in the tool box to go back to the flow line.**

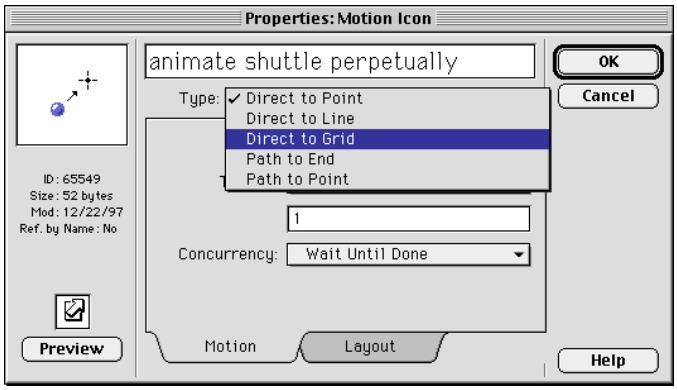


☐ **Double-click on the Motion icon "animate shuttle perpetually."**

Since the shuttle graphic was the most recent thing displayed on the screen, Authorware automatically brings it back up.

We need to change the type of motion to allow for data-driven animation in two-dimensions.

☐ **Click and hold on the words "Direct to Point" to activate the pop-up menu, then select "Direct to Grid":**

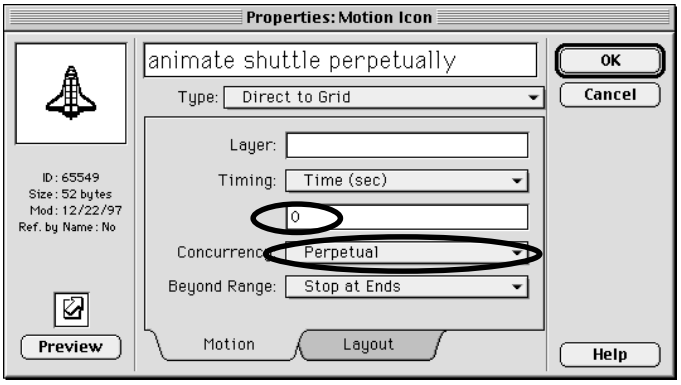


While viewing the “Motion” options, let’s make two more modifications.

If, by chance, you are not viewing the motion options, click once on the “Motion” tab.

☐ **Change the timing to zero (“0”) seconds.**

☐ **Change the concurrency from “Wait Until Done” to “Perpetual”.**

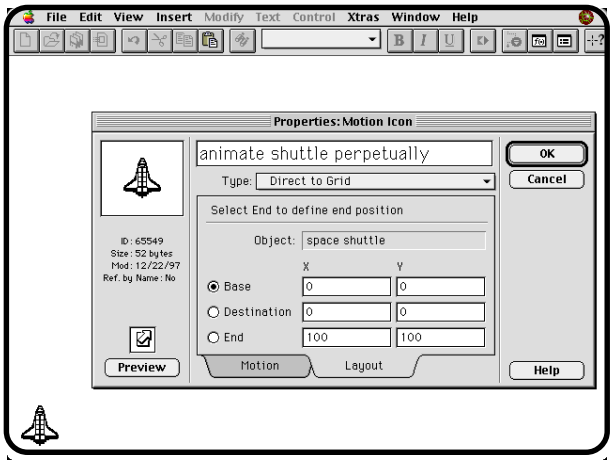


The first modification tells Authorware to update the position of the shuttle instantaneously, in other words, without waiting. (If this sounds mysterious or confusing at this point, don’t worry about it now.) The second modification tells Authorware to animate the shuttle “forever” throughout the rest of the simulation. Authorware will constantly monitor the value of the variables and animate the shuttle accordingly without any additional programming.

The next step is to construct the area in which the animation will take place. These options can be found in the “Layout” part of the dialog box.

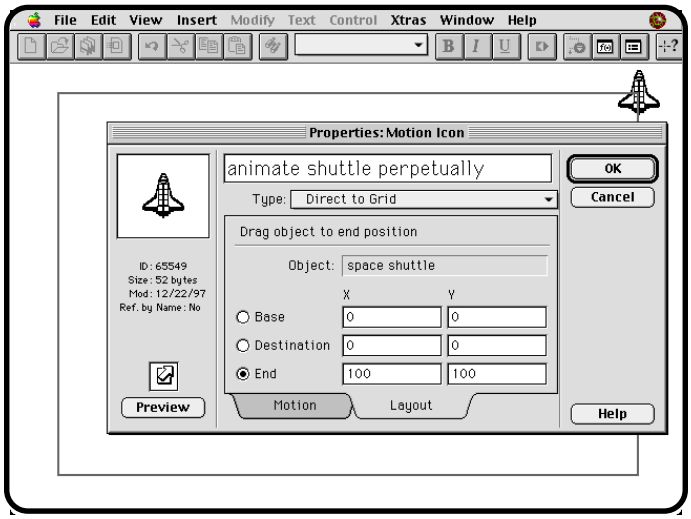
☐ **Click once on the “Layout” tab to view these options.**

☐ **As per the directions in the dialog box, drag the shuttle to the “base position” down in the lower left hand corner of the screen.**



Notice that the directions have changed to match our next step.

☐ **Click on the radio button beside "End," then drag the shuttle to the "end position" up in the top right hand corner of the screen.**

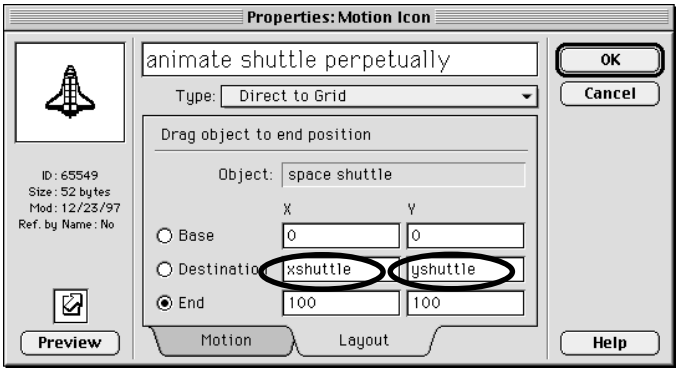


A fuzzy rectangle appears which defines the animation area. Fine tune the position of the animation area as necessary. Remember that you will need to click on either the "Base" or "End" position radio buttons as you fine tune.

Now let's enter the variables that we set up earlier that will control the position of the shuttle (you'll soon see how they work).

☐ **Type "xshuttle" and "yshuttle" in the two text windows beside "Destination" as shown:**

We will let the x,y values for Base remain set to 0 and those for End to 100. These determine the coordinate dimensions for our two-dimensional area. We could change these if we wanted, but these values will work just fine.



By naming our variables "xshuttle" and "yshuttle" it makes it very clear which variable goes in which box. Remember that "x" is the convention for the horizontal dimension and "y" for the vertical.

☐ **When finished, click "OK" to return to the flow line.**

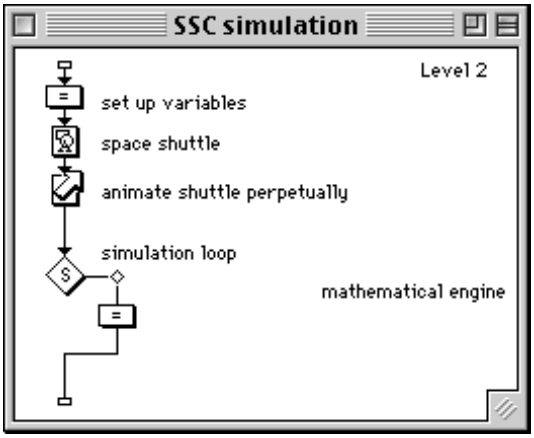
Setting up the simulation's "mathematical engine"

Let's focus our attention now on the "mathematical engine" that will "power" our simulation. The idea here is to build a programming loop that will constantly check the status of the variables "xshuttle" and "yshuttle." As they change, so too will the shuttle's animation.

☐ **Drag a Decision icon to the Level 2 flow line and title it "simulation loop."**

☐ **Drag a Calculation icon to a point just right of the Decision icon and title it "mathematical engine."**

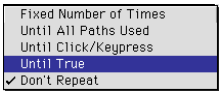
Your Level 2 flow line should look like this:

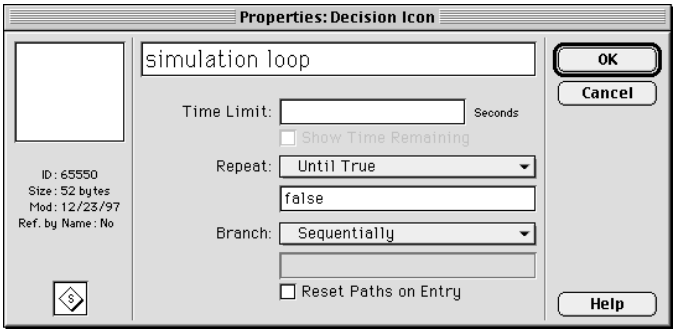


As you can see from the flow line, this Decision icon is set to branch to the Calculation icon, then exits down and out of the Level 2 flow line. We need to set this up as a never ending loop.

☐ **Double-click on the Decision icon "simulation loop."**

☐ **In the dialog box, change the repeat option from "Don't Repeat" to "Until True" in the Repeat group, then type "false" in the text window:**



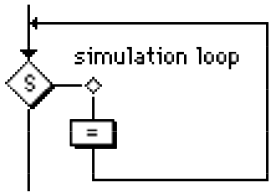


Leave the branching option set to "Sequentially".

Remember, we want this simulation loop to continually repeat itself. The "Repeat Until True" option allows us to set some condition which Authorware checks at the start of each loop. If the condition is true, then Authorware will exit out of the Decision icon's loop. By typing "false" in this box, we guarantee an "infinite loop" since the condition can never be true. Later on, when we build the rendezvous game, we will enter a condition that checks to see if the game has been won. Only then will the simulation loop end. For now, this infinite loop suits our needs just fine.

☐ **Click "OK" to go back to the flow line.**

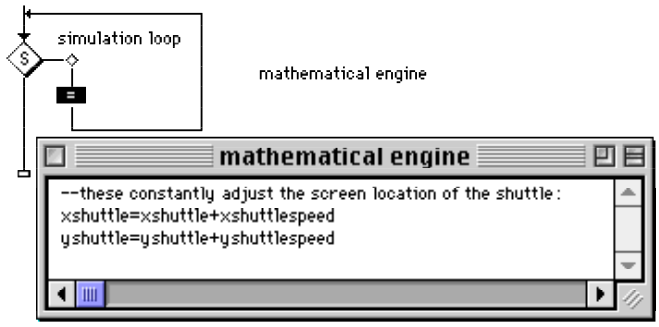
Notice that the flow line has changed to represent the loop we just created:



Next, let's work on the mathematical model underlying this simulation.

☐ **Double-click on the Calculation icon "mathematical engine."**

☐ **Type the following in the Calculation icon's text window:**



The two hyphens at the start of the first line tells Authorware to ignore whatever follows on the line. This is a convenient way to type comments to yourself.

The last two lines are the heart of the engine. Line 2 tells Authorware to "make xshuttle what it was before plus xshuttlespeed." Since xshuttlespeed's initial value is 0, xshuttle will not change. Consider, however, if "xshuttlespeed" was 1 instead. "xshuttle" would get larger by one each loop. As it increases, Authorware would change the position of the shuttle via the Motion icon (recall that the horizontal motion is controlled by "xshuttle"), moving it slowly toward the right of the screen. Line 3 works in a similar way, but for the variable "yshuttle." In a moment we will build the interactive portion of the file which gives the player the opportunity to "nudge" the shuttle around the screen.

☐ **Click on the close box to close the Calculation icon's window, then click "yes" to save the calculation changes.**

☐ **Save the file.**

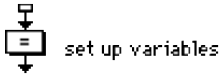
Testing and calibrating the simulation loop and mathematical engine

It's about time we did a little test to see if things are working properly.

☐ **Restart the file.**

If your simulation is working properly, the shuttle should skip from the left of the screen to the center (50,50), and then stay motionless because we are adding 0 each loop to "xshuttle" and "yshuttle."

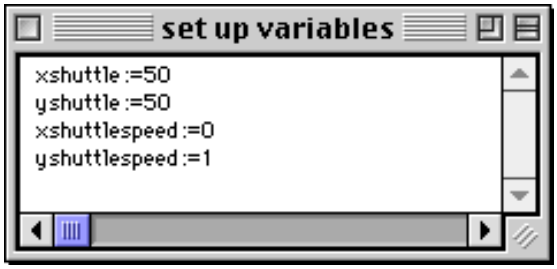
Not very exciting, is it? Let's make one small change to the Calculation icon "set up variables" to test it further.



☐ **Press Command/Control-J to stop the simulation and go back to the flow line.**

☐ **Double-click on the Calculation icon "set up variables."**

☐ **Change the value of "yshuttlespeed" to 1.**



☐ **Click on the close box to close the Calculation icon's window, then click "yes" to save the calculation changes.**

☐ **Restart the file several times.**

Each time you run the file, the shuttle should move toward the top after it skips over to the middle of the screen. When it gets to the top, it will probably stop. Remember, our "universe" is defined by the animation area, so when the shuttle gets to the top, it reaches, in a sense, the end of space!

An important part of the test is to determine an optimum speed for the shuttle based on each loop. This loop speed will depend on the computer you are using (i.e. Macintosh PowerPC will run much faster than a Macintosh LC). For example, I want the shuttle to take about 5 seconds to reach the top of the screen. On my computer, the shuttle took less than one second. Obviously, I need to slow the shuttle down. There are two ways to do so: 1) adjust and test the value of yshuttlespeed (with lower and lower values) until the right speed is reached; 2) go back to the Motion icon ("animate shuttle perpetually") and adjust and test the value of the "End" position (with larger and larger values) until the right speed is reached. We'll opt for the first strategy.

I'm using a Power Macintosh 8500.

☐ **Test different values of yshuttlespeed until the shuttle takes about 5 seconds to reach the top of the screen.**

On my computer, a value of .05 seemed about right.

This concludes the test. Hopefully, your file is working properly. We need to change value of "yshuttlespeed" back to 0 before proceeding. As you see, we will use the value of .05 in the next section when we make the simulation interactive.

Platform Alert!

As mentioned in chapter 1, Macintosh users use the Command key and Windows users use the Control key for all keyboard shortcuts. (No further reminders will be given in this chapter.)

☐ **Open the Calculation icon “set up variables” and change the value of “yshuttlespeed” back to 0; close the icon and choose the save the changes.**

☐ **Save the file.**

Making the simulation interactive

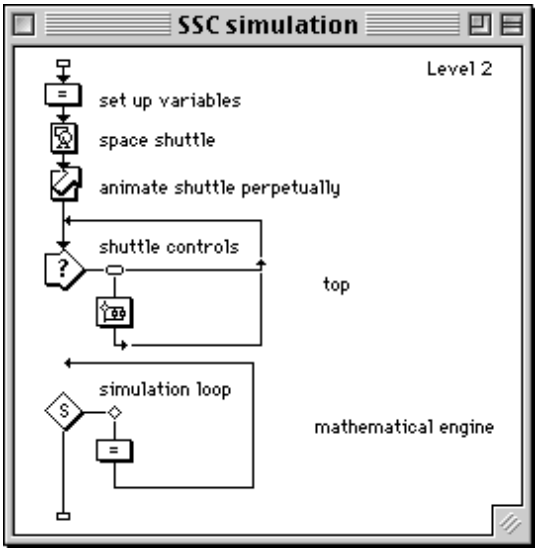
To give the player control over the shuttle, we need to use an Interaction icon. The goal here is to give the player four buttons — top, down, left, and right — which “nudge” the shuttle around the screen according to Newton’s laws of motion.

☐ **Drag an Interaction icon to the Level 2 flow line just below the Motion icon and title it “shuttle controls.”**

☐ **Drag a Map icon to a point just right of the Interaction icon.**

☐ **Make sure it is a “button” response type, then click “OK” in the response type dialog box.**

☐ **Title this Map icon “top.”**



As you can see, the flow line is broken between the “shuttle controls” Interaction icon and the “simulation loop” Decision icon. As of now, the simulation waits until the player presses the button “top” and then continues back around to the “shuttle controls” icon. As is, the flow will *never* get to the simulation loop. We need to make the simulation con-

tinue looping around the mathematical engine *while* giving the player the opportunity to nudge the shuttle. Fortunately, Authorware gives us the option of creating *perpetual* interactions.

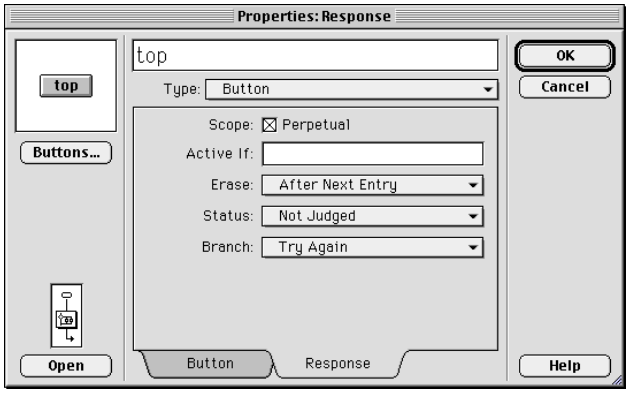
Think of this loop as "running" the engine. (Fortunately, our engine runs on electricity instead of gasoline!)

☐ **Double-click on the baby pushbutton just above the branch leading into the Map icon "top."**

This brings up the dialog box for the button options. Most probably, the "Button" options are showing. The perpetual option is located in the "Response" options.

☐ **Click on the "Response" tab to view these options.**

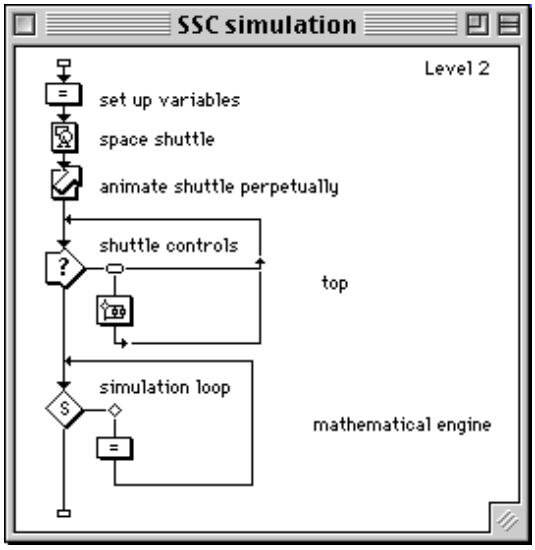
☐ **Click once in the small box beside "Perpetual."**



☐ **Click "OK."**

Notice how the flow line has now changed:

Some might argue that it is also better to change the flow of this branch to "Exit Interaction" or "Return" instead of "Try Again." However, there is no perceptible difference in the execution of the program, so we will leave it as is for simplicity. (My personal preference would be "Return", an branching option only available for perpetual interactions that takes the user back to whatever point on the flow line they were before clicking the perpetual button — this is a very useful option for embedding features such as glossaries and other resources in tutorials.)



The flow line is now joined between the Interaction icon and the Decision icon. This reflects the fact that the button

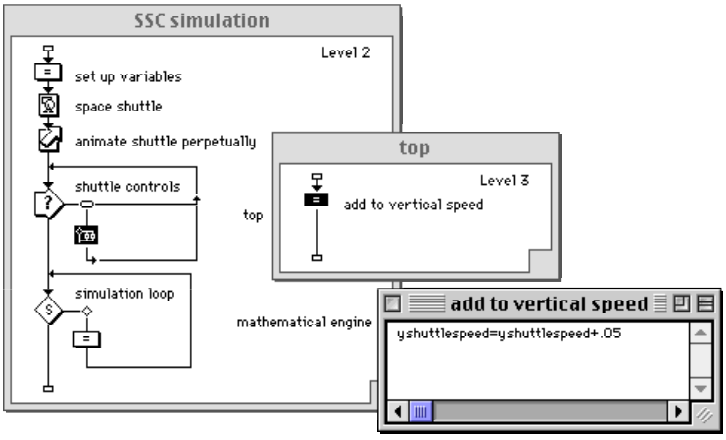
"top" is perpetual. Instead of waiting indefinitely for the player to press the "top" button, the flow immediately goes on to "simulation loop." However, Authorware continues to "wait and watch" for the user to press the top button. When pressed, the flow is instantly "transported" to the point indicated by the baby pushbutton. Whatever is inside the Map icon "top" is then executed and the flow continues out of the Map icon, back around and finally down to "simulation loop" until the next time the perpetual button "top" is pressed.

Let's put something inside the "top" Map icon that will change the motion of the space shuttle.

☐ **Double-click on the Map icon "top."**

☐ **Drag a Calculation icon to the Level 3 flow line and title it "add to vertical speed."**

☐ **Double-click on this Calculation icon and type "yshuttlespeed=yshuttlespeed+.05" in the text window:**



This line tells Authorware to "make yshuttlespeed what it was before plus .05." Recall that "yshuttlespeed" has an initial value of 0. If we press this button once, it increases to .05. Also recall that this value is constantly added to "yshuttle" in the mathematical engine and that "yshuttle" is used in the Motion icon to control the animation of the shuttle. Therefore, when "yshuttlespeed" is .05, the shuttle will animate slowly to the top. If we click on the button again, "yshuttlespeed" will increase to .1 and the shuttle will move twice as fast.

Why .05? Recall that this was the value that I settled on when testing the simulation back on pg. 175.

☐ **When finished, click on the close box to close the Calculation icon's window, then click "yes" to save the calculation changes.**

Let's give it a try.

☐ **Restart the file.**

The shuttle skips over to the center of the screen and "waits."

☐ **Click once on the "top" button.**

The shuttle begins to move to the top. Again, it stops when it reaches the top "edge of the universe."

☐ **Restart the file again.**

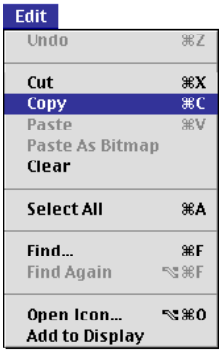
☐ **Click the "top" button twice.**

The shuttle will move twice as fast to the top (it will again stop when it reaches the top).

Pretty nifty! But let's go ahead and quickly set up another button that we can use to nudge the shuttle back to the bottom.

☐ **Press Command/Control-J to jump back to the flow line.**

We want a second button that is very similar to the button "top." It also needs to be perpetual and it also needs to change the motion of the shuttle. Here is where the "copy and paste" feature can save us lots of time. We will copy and paste the Map icon "top" and then make some small (but important) changes.



Before you can copy something, you have to first select it.

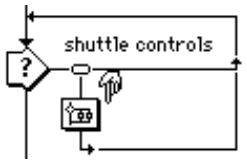
☐ **Click once on the Map icon "top."**

☐ **Choose "Copy" from the "Edit" menu.**

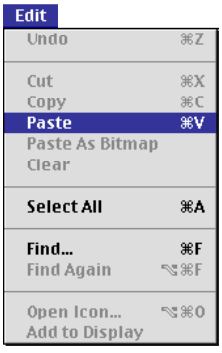
This icon and all of its contents and "qualities" (such as th- perpetual feature) are now on the clipboard

paste hand

☐ **Click once just to the right of the Map icon "top" so that the "paste hand" appears:**



top

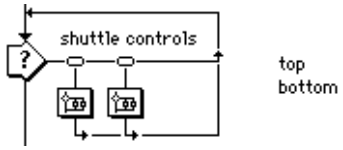


☐ **Choose "Paste" from the "Edit" menu.**

Icons previously cut or copied are pasted to wherever the paste hand is pointing.

You now have two buttons branches titled "top."

☐ **Rename the second branch "bottom":**

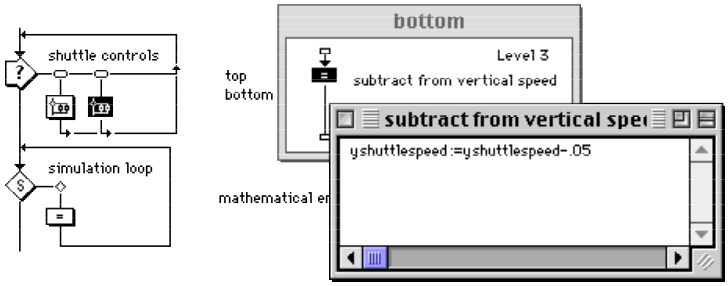


Now let's go inside the Map icon "bottom" to change the name of the Calculation icon and then make one small but incredibly important change to the programming.

☐ **Double-click on the Map icon "bottom."**

☐ **Change the title of the Calculation icon to "subtract from vertical speed."**

☐ **Open the Calculation icon and change the "+" (plus sign) to "-" (minus sign):**



Why change from a plus sign to a minus sign? Think about it. Given the way we set up the mathematical model, numbers get bigger as we go from bottom to top (from 0 to 100) and get smaller as we go from top to bottom (from 100 to 0). When "yshuttlespeed" is 0, the shuttle is not moving at all vertically. When "yshuttlespeed" is a positive number, the shuttle moves from bottom to top. It will move faster the bigger "yshuttlespeed" gets. When "yshuttlespeed" is a negative number, the shuttle will move from top to bottom. Why? Here is the line of programming in "mathematical model":

```
yshuttle:=yshuttle+yshuttlespeed
```

When "yshuttlespeed" is negative (say -2), we are adding a negative number to "yshuttle." This is the same algebraically as subtracting.

☐ **When finished, click on the close box to close the Calculation icon's window, then click "yes" to save the calculation changes.**

This sends us back to the flow line. Time to test the simulation.

☐ **Restart the file.**

Click the "top" button, wait until the shuttle gets close to the top (but not all the way), then click the "bottom" button.

The shuttle should come to a dead stop. Continue testing how the buttons work together. One warning: try to stay away from the edge of the "universe," that is, the edges of the animation area. We'll explain later what happens when the shuttle actually reaches these special edges plus change the simulation so that the shuttle "wraps around" like a video game — when it goes off of the top it will reappear at the bottom and visa versa.

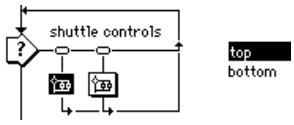
☐ **When finished testing, press Command/Control-J to jump back to the flow line.**

☐ **Save the file.**

Finishing the interaction

Let's add the final two branches, "left" and "right," that will enable the player to also nudge the shuttle across the screen horizontally. Again, we will use the copy and paste feature to speed up the process.

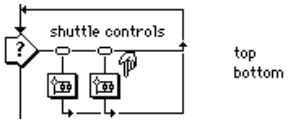
☐ **At the flow line, click once on the Map icon "top."**



☐ **Choose "Copy" from the "Edit" menu.**

Again, this copies the contents of the branch and its "qualities" (such as the perpetual feature) to the clipboard.

☐ **Click once just to the right of the Map icon "bottom" so that the "paste hand" appears.**

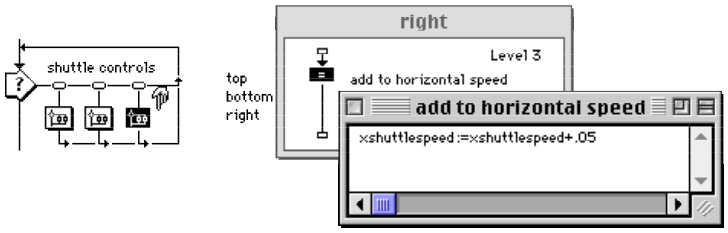


☐ **Choose "Paste" from the "Edit" menu.**

☐ **Rename this third branch "right."**

☐ **Double-click to open the Map icon "right" and rename the Calculation icon "add to horizontal speed."**

☐ **Double-click to open this Calculation icon and change the line of programming as per the following:**



As you can see, the variable is changed to "xshuttlespeed" but the programming logic is the same.

☐ **When finished, close the Calculation icon's window and click "yes" to save the calculation changes.**

☐ **Close the Level 3 window.**

Just one more branch to go.

☐ **Click once on the Map icon "right" (the third branch).**

☐ **Choose "Copy" from the "Edit" menu.**

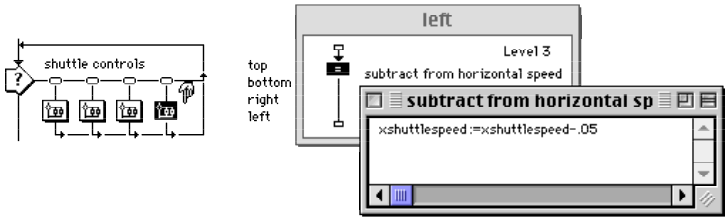
☐ **Click once just to the right of the "right" branch so that the paste hand appears.**

☐ **Choose "Paste" from the "Edit" menu.**

☐ **Rename this Map icon as "left."**

☐ **Double-click to open the Map icon "left" and rename the Calculation icon "subtract from horizontal speed."**

☐ **Double-click to open this Calculation icon and change the "+" (plus sign) to a "-" (negative sign):**



☐ **When finished, close the Calculation icon's window and click "yes" to save the calculation changes.**

☐ **Restart your file several times to test the simulation.**

Each button should "nudge" the shuttle in its direction. Again, try to stay away from the edges. Notice how you have to carefully balance nudges on the horizontal (left with right) or vertical (top with bottom) plane to get the shuttle to stop.

If you understand why one nudge to the right followed by one nudge to the left makes the shuttle move and then stop, then you have just begun to understand more deeply the meaning behind Newton's laws of motion, specifically his second law which states that "force equals mass times acceleration," abbreviated as "f=ma".

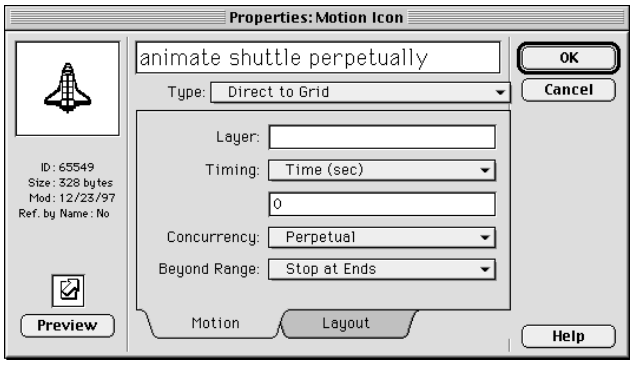
☐ **Press Command/Control-J to jump back to the flow line.**

☐ **Save your file.**

Creating a "wrap-around" universe

So far, we have been careful to avoid the edges of the animation area as we have tested the simulation. Let's explain why. Recall that the animation area goes from 0 to 100 both horizontally and vertically. When "xshuttle" or "yshuttle" gets larger than 100, Authorware has a dilemma. What should it do?

If you open the Motion icon and look inside the Layout options, you will see that Authorware is set by default to "Stop at Ends."

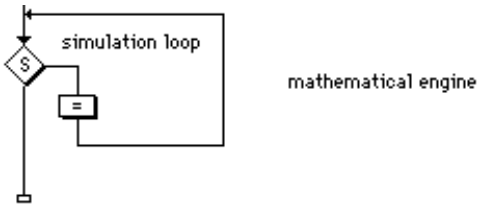


This means just what is says — when the shuttle gets to the edge, Authorware keeps it at the boundary, even though the variable controlling it continues to increase. So, in a sense, the top and right boundaries can be interpreted "anything 100 or above" and the left and bottom boundaries as "anything 0 or below." The shuttle stays at the boundary until the variable finally gets back to within the range of 0-100.

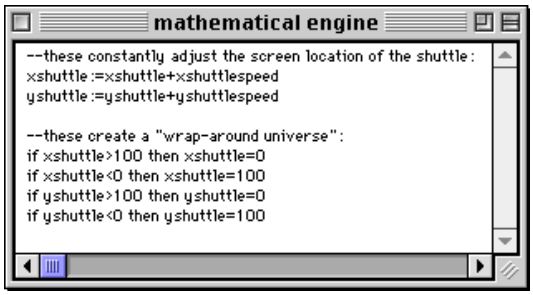
We want a "wrap around universe," similar to most video games — when the shuttle goes off the top it simply reappears at the bottom.

The simplest way to get a "wrap around universe" is to change the Motion icon from "Stop at Ends" to "Loop." For example, when the variable "yshuttle" goes above 100, Authorware resets the animation area to 100 to 200. If it goes below 0, it resets the animation area to -100 to 0. Frankly, for our purposes, this would work just fine. However, we are going to take advantage of the problem to learn a little programming trick that *resets the variable* when it crosses the boundary. Let's do it.

☐ **Double-click on the Calculation icon "mathematical engine."**



☐ **Add the following five lines to the text window (resize the window if necessary):**



As you can see, the last four lines use the "IF-THEN" function to reset our two engine's variables. These programming lines tell Authorware at the end of *each* loop to check if either variable is "out of bounds" and to reset them to the opposite boundary if they are.

The IF-THEN function is in the "language" category of functions.

☐ **When finished, close the Calculation icon's window and click "yes" to save the calculation changes.**

☐ **Restart your file to test the simulation.**

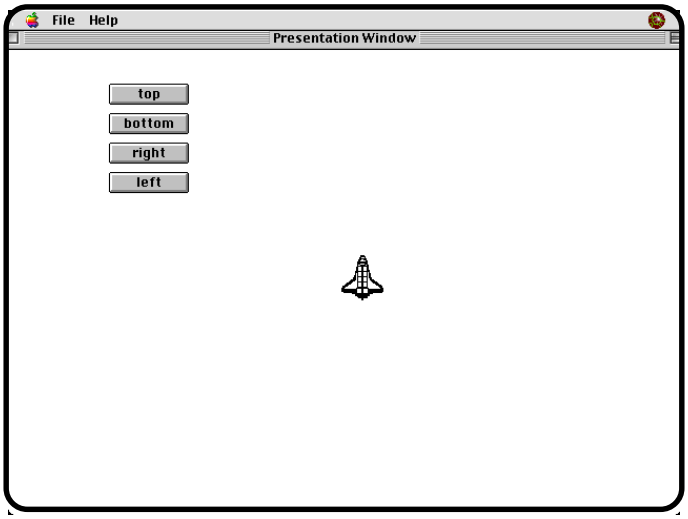
You no longer need to worry about the edges of the "universe."

☐ **Press Command/Control-J to jump back to the flow line.**

☐ **Save your file.**

Improving the user interface

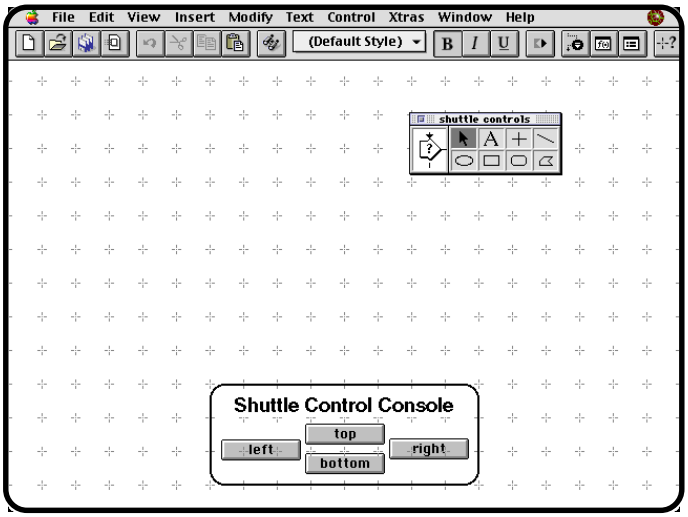
Up to this point, your screen probably resembles the following as your run the simulation:



This is terrible! Although we might want to later make some big changes so that it resembles the "cockpit" of the space shuttle, let's make some small changes now that will drastically improve the usability of the simulation (even for us).

☐ **At the flow line, double-click on the Interaction icon "shuttle controls".**

☐ **Change the display so that it resembles the following:**



Arranging the buttons in this way is an example of "natural mapping," a powerful idea in user interface design.

☐ **When finished, click in the close box in the tool box to go back to the flow line.**

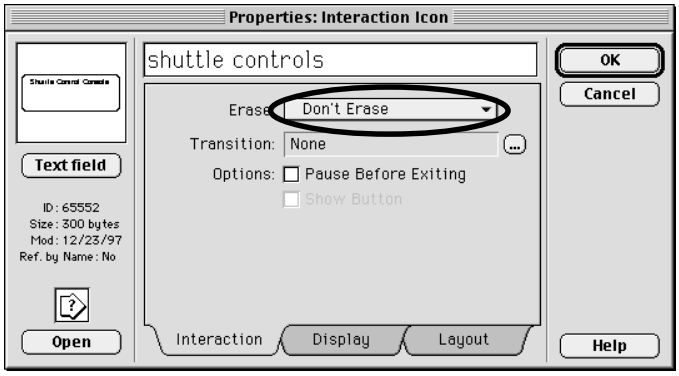
☐ **Restart your file to test it.**

Unfortunately, the console graphic will quickly disappear. The reason is that the automatic erasing option for this Interaction icon is still set to the default value of "Upon Exit." Therefore, we need to change this to "Don't Erase." As introduced in previous chapters, you can access the Properties dialog box for an Interaction icon two ways: 1) click once on the icon to select it, choose "Icon" from the "Modify" menu, then select "Properties..."; or 2) hold down the Command/Control key and then double-click on the icon. As before, we'll do the latter.

☐ **Press Command-J to go back to the flow line.**

☐ **While holding down the Command (Macintosh) / Control (Windows) key, double-click on the Interaction icon "shuttle controls".**

☐ **Be sure you are viewing the Interaction options and choose "Don't Erase" from the pop-up menu:**



☐ **Click "OK".**

☐ **Restart your file several times to test it.**

☐ **Press Command/Control-J to go back to the flow line.**

☐ **Save your file.**

Congratulations! You now have a working prototype of a physics simulation of Newton's laws of motion.

This concludes the first session of this chapter. Next, we will add gaming features to the simulation.

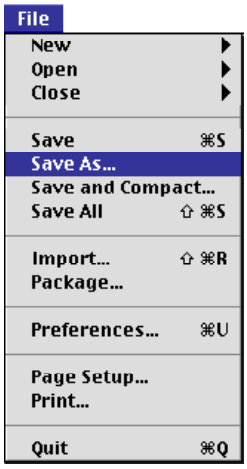
Adding a game context to the simulation: Rendezvous

The simulation works! It's neat! It has all kinds of educational potential! But left as it is your middle school students will probably get bored with it quickly. One idea is to add some gaming features to it. No, we are not trying to compete with Nintendo here, but rather to give the students a more motivating context for exploring the simulation and, most importantly, a context for goal setting and evaluating their understanding.

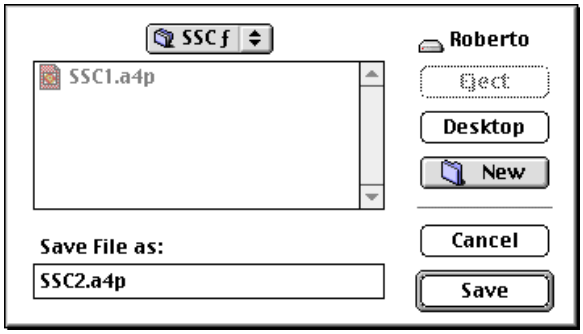
The goal of the game is to rendezvous, or dock, the shuttle with a space station somewhere deep in space. It might be a good idea to keep a working copy of the simulation as it now stands, so we will start by saving another copy of the simulation with the name "SSC2.a4p".

☐ **Choose "Save as..." from the "File" menu:**

Another way to copy the file is to quit Authorware, go back to the desktop, choose to "duplicate" the file "SSC1.a4p," change its name to "SSC2.a4p," then double-click on it to launch Authorware and load the file.



☐ **Change the file name to SSC2.a4p, select an appropriate destination, then click "Save":**



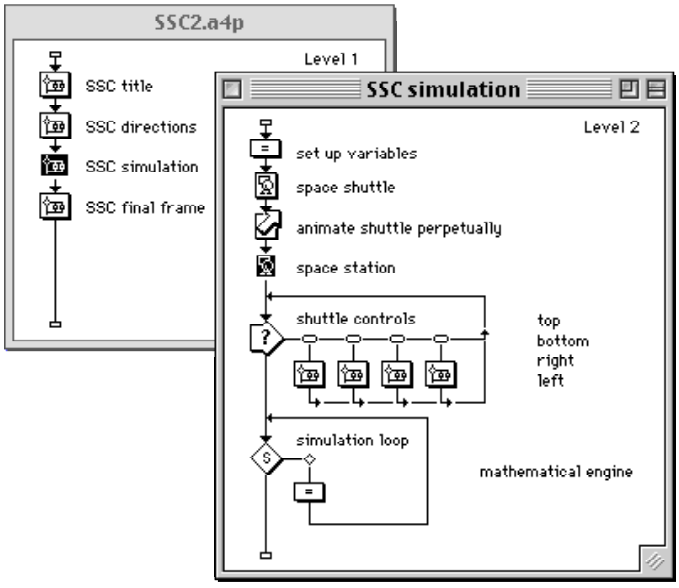
You are now working with a completely separate file. SSC1.a4p is “safe and sound” and will remain unchanged from here on.

Actually, adding a game context to the simulation involves surprisingly few steps. We will need to create a space station, then tell Authorware to keep checking whether or not the shuttle and the space station overlap.

Creating a space station

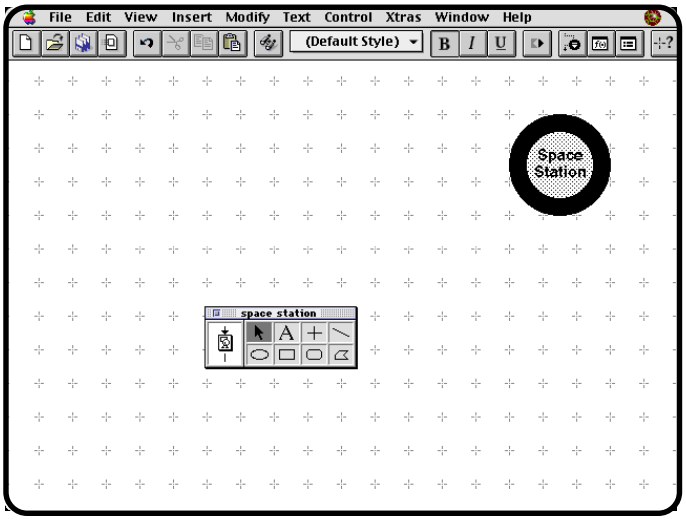
The first step is to create a space station. Again, we’ll keep the graphics simple so as to move quickly through the process.

☐ **At the Level 2 flow line (within the Map icon “SSC simulation”) drag a Display icon just below the Motion icon “animate shuttle perpetually” and title it “space station”:**



☐ **Double-click on the Display icon “space station.”**

☐ **Construct a suitable "space station" graphic in the top right-hand portion of the screen.**

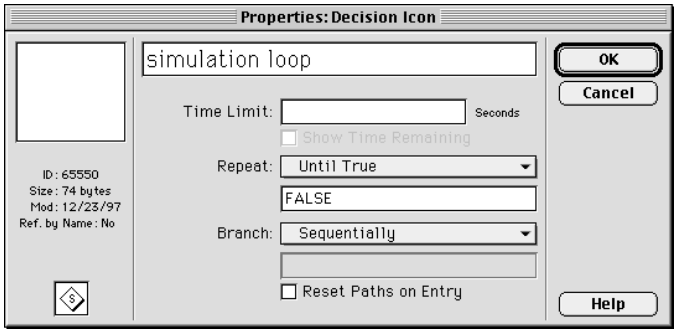


☐ **When finished, click in the close box in the tool box to go back to the flow line.**

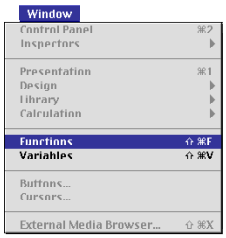
The goal of the game is to maneuver the shuttle to the space station. We need to figure out a way to tell Authorware to keep checking as to whether the shuttle has made a successful rendezvous with the space station. We will explore two strategies. The first is the simplest and involves a nifty graphics function built in to Authorware — the "overlapping" function.

☐ **Double-click on the Decision icon "Simulation loop."**

Remember this dialog box?



Recall that we typed in the word "FALSE" into the condition window under "Repeat Until True" to make an infinite loop. We are now going to replace this with a condition based on whether the shuttle graphic and space station graphic are overlapping.



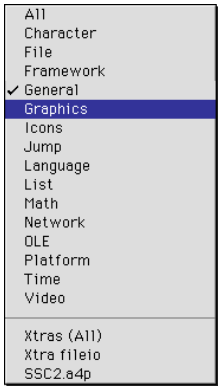
☐ **Delete the word “FALSE” and be sure that the cursor is flashing inside the “Until TRUE” text entry box.**

☐ **Choose “Functions” under the “Window” menu.**

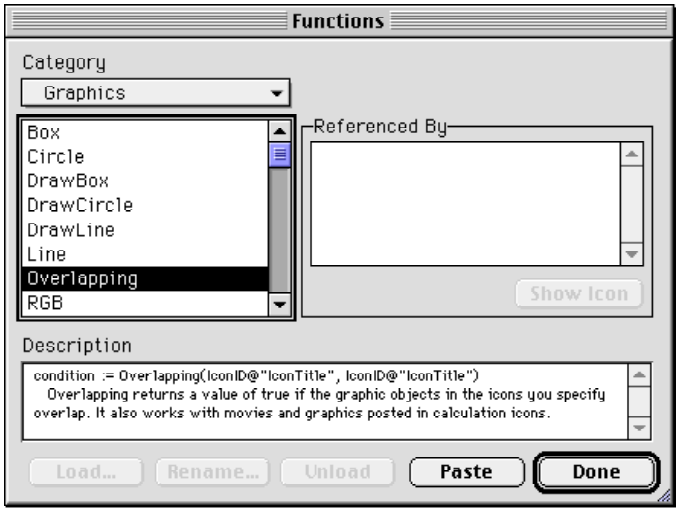
Toolbar Shortcut!

Click here

☐ **Change the pop-up category menu to “Graphics.”**



☐ **Scroll down and click once on the function “Overlapping”:**

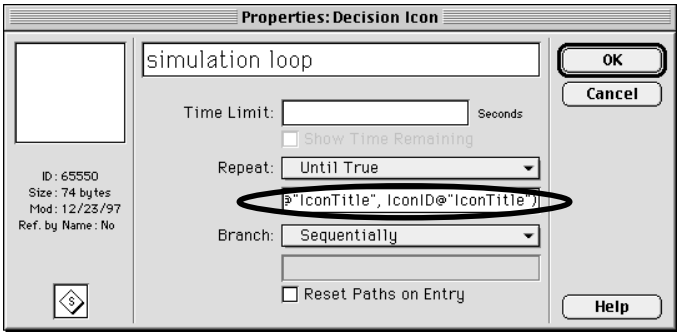


☐ **Click “Paste”.**

This pastes the function, in its generic form, into the text entry window (the window may be hidden at this point).

☐ **Click “Done.”**

This is what you will probably see:



Understand that this is a rather long function which doesn't fit conveniently in the text window. However, if you press the left and right arrow buttons on the keyboard, you will see that the following has actually been pasted:

```
Overlapping(IconID@"IconTitle", IconID@"IconTitle")
```

All we need to do is enter the titles of the two displays we want Authorware to check.

☐ **Replace the first 'Icon Title' with 'space shuttle' and the second with 'space station.'**

You can also use the mouse to maneuver inside this text window, but it takes patience and dexterity.

Again, you may need to use the left and right arrow keys on the keyboard to get to the right spots. You must type the display titles exactly as they are on the flow line. If you don't, Authorware will "scold" you and prevent you from closing this dialog box. When finished, it should be typed exactly as the following:

```
Overlapping(IconID@"space shuttle", IconID@"space station")
```

This assumes, of course, that these are also the names you used to title these icons. If not, be sure to use your titles here.

☐ **When finished, click "OK."**

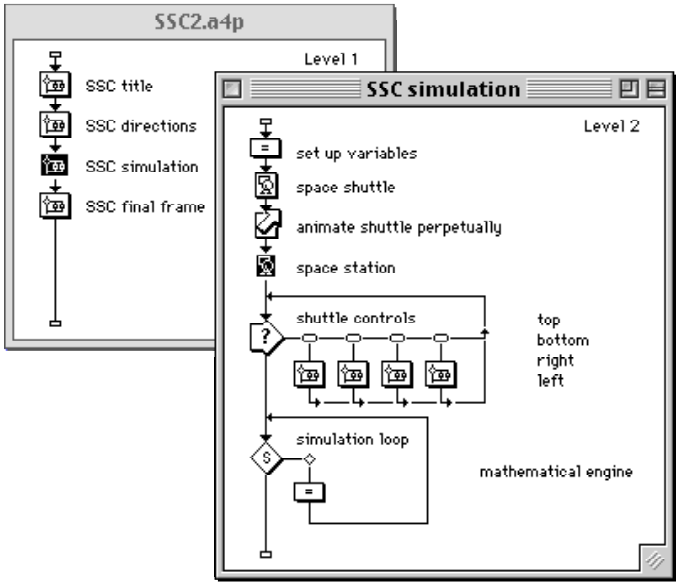
Authorware checks to see if there are two displays with these names. If it finds them, it closes the dialog box. Otherwise, you will get an error message like the following:

What's wrong here? Notice that "shuttle" is misspelled.

☐ **Restart your file to test it.**

Your simulation should run just like before except that the simulation will abruptly stop as soon as the shuttle comes in contact with the space station.

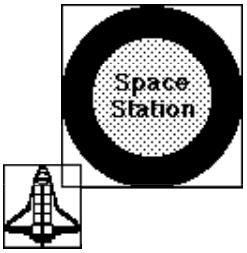
The reason for this can be found by examining the flow line:



The simulation loop runs continuously until the overlapping condition is true, at which point it exits the Decision icon loop. The flow goes to the bottom of the Level 2 flow line which is equivalent to exiting the "SSC simulation" Map icon on the Level 1 flow line. The flow then enters the "SSC final frame" Map icon. Since there is nothing as yet in this map, the flow just continues to the bottom of the Level 1 flow line which is equivalent to the end of the file — so the file just stops.

You also have probably noticed at this point that the shuttle does not have to actually overlap the space station for the condition to be triggered, but merely get very close. The reason for this is that the overlapping function has one major limitation. It only checks to see if the rectangular background area in which the displays reside are overlapping, not the foreground images.

If this is confusing, just imagine "encasing" each graphic in a rectangle:

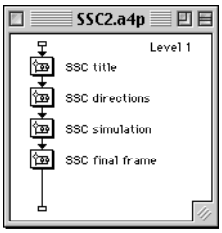


In this example, Authorware "sees" these as overlapping even though it looks pretty clear to us that they are not. There is a solution to this problem that involves some mathematics, but we'll save that for later.

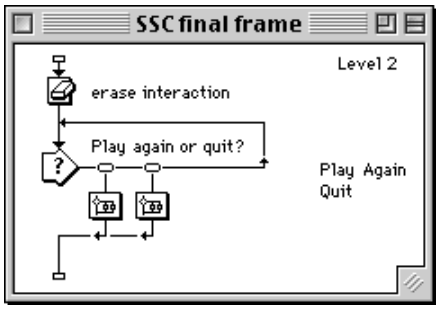
Constructing the "final frame"

Next, let's construct a final frame that's identical to what we did in chapter 3 except that the message will be different (in fact, you could just copy and paste the final frame Map icon from that file into this one, then modify the Erase icon and display of the Interaction icon).

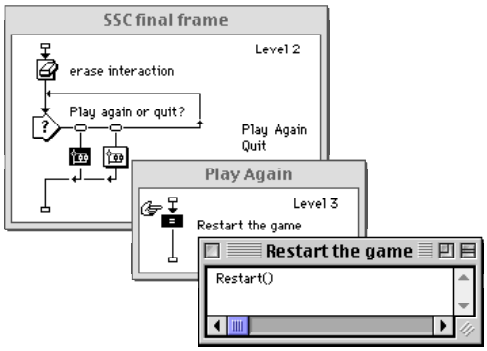
Here are the abbreviated instructions (see chapter 3 for the "long version"):



- ☐ **Double-click the Map icon "SSC final frame."**
- ☐ **Drag an Erase icon to the Level 2 flow line and title it "erase interaction."**
- ☐ **Drag an Interaction icon to the Level 2 flow line and title it "Play again or quit?".**
- ☐ **Continue constructing the interaction so as to have two pushbutton branches, one title "Play again" and the other "Quit."**

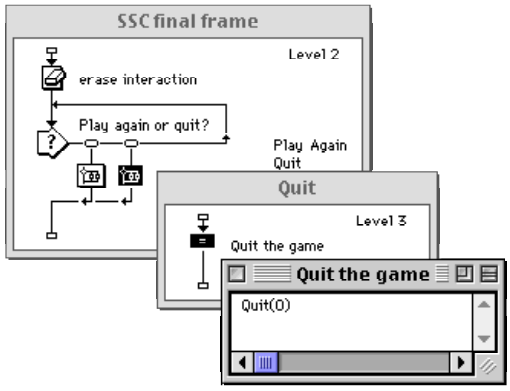


- ☐ **Open the "Play again" Map icon, drag a Calculation icon to the Level 3 flow line, and title it "Restart the game."**
- ☐ **Type the function "Restart ()" into this Calculation icon, then close this icon.**



☐ **Open the "Quit" Map icon, drag a Calculation icon to the Level 3 flow line, and title it "Quit the game."**

☐ **Type the function "Quit (0)" into this Calculation icon, then close this icon.**



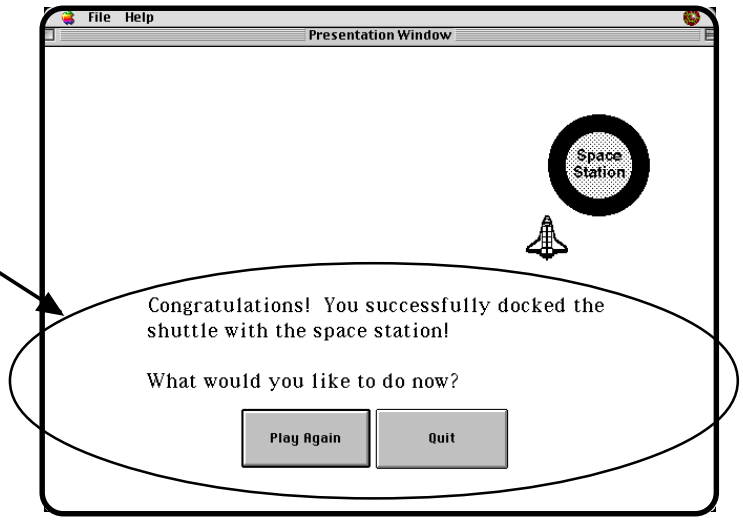
☐ **Restart the file.**

☐ **Maneuver the shuttle to the space station to win the game; when the Erase icon appears click on the display of the shuttle control console and the four perpetual buttons to erase them, but do not erase the shuttle and space station.**

This will keep the shuttle and space station graphics on the screen — a nice touch when displaying the following note of "congratulations."

☐ **Construct the display for the Interaction icon "Play again or quit?" to match the following:**

This is the part you should construct.



☐ **Restart the file to make sure it works properly.**

☐ **Save the file.**

A mathematical approach to determining if the displays overlap

My apologies if this section reads mostly like a math lesson (but that probably best describes it).

I wish I had this example to work on when I was studying the distance formula in high school. I might have paid attention!

It's clear that the built-in overlapping function of Authorware is not satisfactory. Here is another approach that uses something called the distance formula from mathematics. It's a *very* useful math idea for anyone interested in game design. However, this is a section you may safely skip for now (so long as you don't mind the way in which Authorware's overlapping function works).

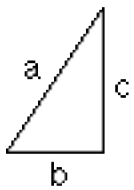
Although you may not remember it, you almost certainly learned the distance formula in high school. Here it is in its traditional form:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

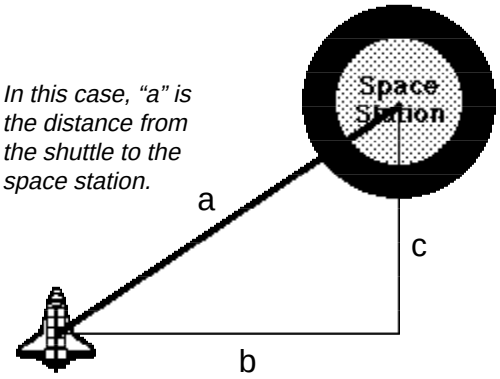
where (x_1, y_1) and (x_2, y_2) define two points on a two-dimensional plane and d is the distance between them.

This formula, though it looks terribly complex, is just a variation of the Pythagorean theorem in which the square of the length of a right triangle's hypotenuse is equal to the sum of the squared lengths of the other two sides:

$$a^2 = b^2 + c^2$$



In our case, we need to know the distance between the space shuttle and the space station:



In this case, "a" is the distance from the shuttle to the space station.

Notice that we have to identify each graphic as one point at its center.

You might be saying at this point that we do have the position of the shuttle with the variables "xshuttle" and "yshuttle." However, these variables are only useful within the arbitrary animation area that we set up. The space station does not "exist" there. DISPLAYX and DISPLAYY use the screen itself as the area and each uses the pixel as the "unit of measurement."

If we had the exact screen locations of both graphics, we could use the distance formula (a version of which Authorware understands) to figure the distance between them. Then, we could set whatever minimum distance we wanted as the trigger for the condition in the Decision icon "simulation loop."

Fortunately, Authorware has two system variables — DISPLAYX and DISPLAYY that monitor the horizontal and vertical screen location of all graphics contained in a Display icon. This location is given as a single point for *all* the graphics in the icon. Authorware identifies this point as the center of the rectangular background of the graphic, such as the following:

The point at which the two diagonal lines cross is the screen location of this graphic as computed by Authorware with the DISPLAYX and DISPLAYY system variables.

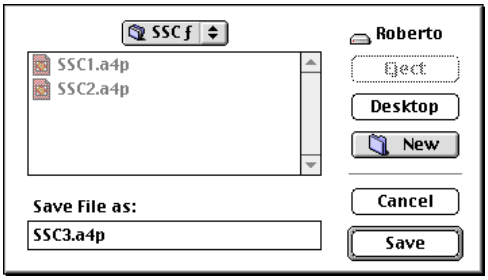
If there is more than one graphic in the icon, Authorware finds the one center point for the group.

We need to create five more variables: two for the screen location of the shuttle, two for the screen location of the space station, and one for the distance between them.

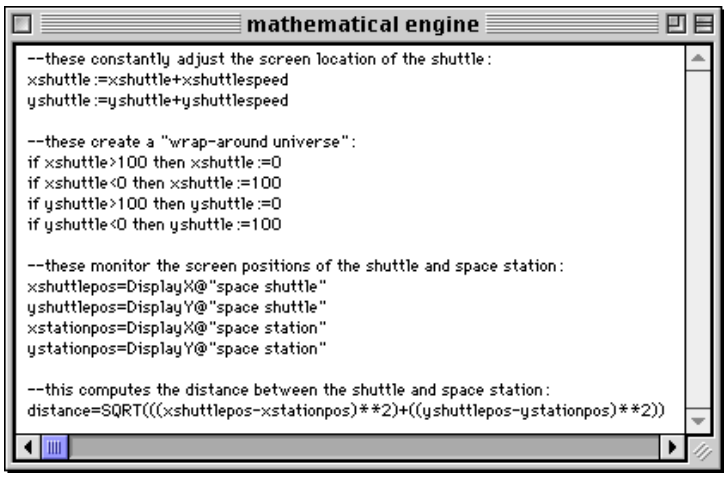
Before we begin making the modifications, let's change the name of the file to "SSC3.a4p." This way, you can refer back to SSC2.a4p to review the method using the overlapping function.

☐ **Select "Save as..." under the "File" menu.**

☐ **Change the file name to SSC3.a4p, select an appropriate destination, then click "Save":**



☐ **Open the Calculation icon "mathematical engine" and add the following lines:**



You can learn more about the DISPLAYX and DISPLAYY system variables by choosing "Variables" under the "Window" menu. Both are in the category "icons."

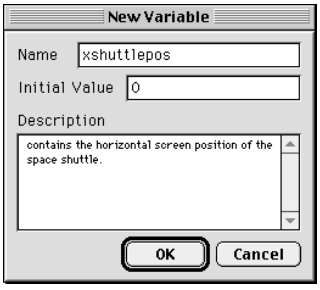
Holy cow! What a line! It really is just the distance formula as described earlier. Type it in carefully — each character counts.

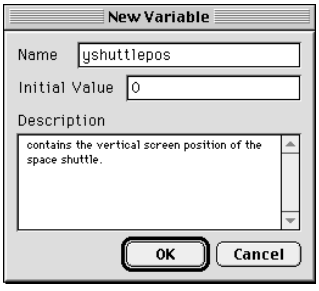
The last line is the distance formula, translated so that Authorware can understand. The notation "**2" is how you tell Authorware to square some quantity (i.e. raise it to the power of 2). Notice the four sets of parentheses. Three sets are used to group the various subparts together and one is used with the SQRT function to take the square root of the whole thing.

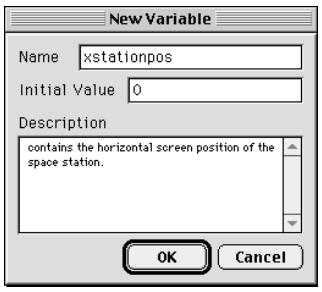
☐ **When finished, close the Calculation icon and choose to save the changes to this icon.**

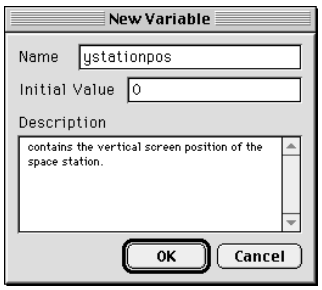
You will need to define the new variables.

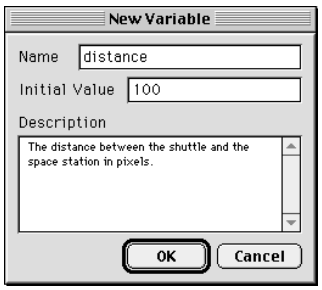
☐ **Type in the following initial values and descriptions for each of the following new variables, and click "OK" after each one:**







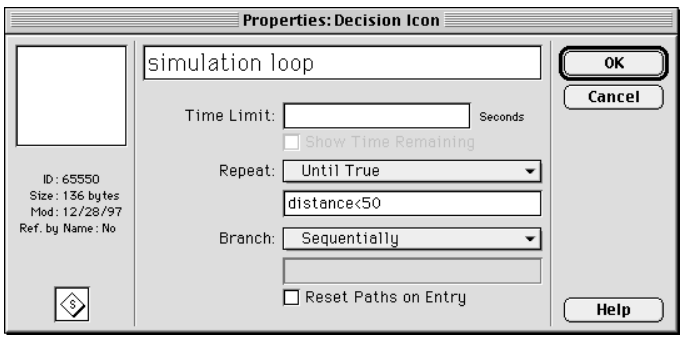




Notice that the initial value of "distance" is set to 100. The reason is that we want to pick a number that will be safely greater than the minimum distance used to trigger the condition in the Decision icon (see below).

The last thing we have to do is change the condition inside the Decision icon "simulation loop" to trigger the end of the simulation depending on the distance between the space shuttle and the space station, currently held by the variable "distance."

☐ **Double-click on the Decision icon "simulation loop" and change the "Repeat Until True" condition to the following:**



This means that Authorware will repeat the simulation loop until the distance between the space shuttle and the space station is less than 50 pixels. This number was chosen by

trial and error. You may want to try different values here after you try out the simulation.

☐ **When finished, click "OK."**

☐ **Restart the file.**

You'll notice that you now need to definitely need to get the shuttle to touch the space station before getting congratulated.

☐ **Save the file.**

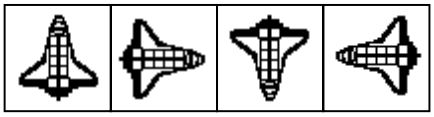
☐ **Package the file.**

See Appendix A for background and instructions on packaging.

Next steps

The "arcade-like" feel to this game points to all kinds of possible modifications. Probably the most obvious is to place the space station at random spots at the start of each round. To do this, you must set up a separate animation for the space station (using the Motion icon) that uses another pair of variables to control the horizontal and vertical position of the space station on the screen. Each of these two new variables would be generated with the Authorware's random function (such as illustrated in chapters 2 and 3). You could also have the space station move around the screen (making for a "moving target"). The game could also be made more challenging by having space stations of different sizes (or by requiring the player to maneuver their shuttle closer to the center of the space station). You could also require the player to bring their shuttle to a dead stop inside the station before being counted as a successful rendezvous.

Some other obvious next steps are not as easy to explain quickly. For example, most people soon wonder how to have the computer rotate the shuttle to show its current trajectory. Unfortunately, Authorware does not have any such rotating function. However, it is possible to create animated "movies" with other applications, such as Macromedia Director or Adobe Premier, in which the shuttle is shown rotated 90 degrees, one rotation per frame:



In Authorware, you can then control which frame is shown at any one time with yet another variable. Obviously, if you wanted to show more rotations, such as in 45 degree increments, you have that many more frames. Chapter 6 shows

how to manipulate such a movie to get the illusion of rotation.

Of course, interesting questions pop up from mathematics and physics when you do this. For example, imagine giving the shuttle one "kick" if it were first pointing at a 45 degree angle. How much of this kick (i.e. force) should be given to the horizontal variable and how much to the vertical variable? In our simulation, we give either all of it (1) or none of it (0). Therefore, you might be tempted to say .5 because that would be half. The actual answer (rounded off) is .707 (for *both* the horizontal and vertical variables). If the shuttle were pointed at a 30 degree angle, the horizontal variable would get approximately .866 of the kick and the vertical variable would get exactly .5 of the kick. Where are these numbers coming from? Once again, the answer comes from your high school mathematics classes (trigonometry). The horizontal variable's share turns out to be the cosine of the angle and the vertical variable's share is the sine of the angle. If you study this issue just a little bit further, you'll find that these answers can also be derived from the good old Pythagorean theorem.

Once again, I contend that designing a computer game is a much more relevant and meaningful reason to learn algebra and trigonometry.

Summary

In this chapter you created a physics simulation in which an animated object (the space shuttle) obeys Newton's laws of motion. You learned about Authorware's overlapping function and its limitations. You learned how to use the distance formula as a mathematical approach to improving the game. You learned how to create an interaction built solely on perpetual buttons which allow the player to interact with the simulation even while it is running a mathematical "engine" modeled on these physical laws. You programmed Authorware to create a "wrap-around universe," similar to many video games.

Other projects

1. Change the "wrap-around universe" to a bouncing universe.

2. Change the context of the simulation from docking the space shuttle with a space station to something out of science fiction (such as Star Wars) just by replacing the shuttle and space station graphics.

3. Instead of ending the game when the shuttle docks with the space station give the player a certain number of points.

Chapter 5: Amazing Mazes

Building a Maze Game

This chapter shows how to create a game in which players maneuver a game piece through a maze as fast as they can. The player's score is based on how long it takes to get through. The player encounters gates at points along the way at which they have to answer a question from history. If they answer correctly, the gate opens and they are allowed to pass through. If they answer incorrectly, they have to go back to the starting point.

In this session you will be saving the following file:

maze.a4p

This chapter assumes you are familiar with the structure of Authorware files, the use of the flow line, and the Map, Display, Erase, Interaction, Calculation, and Motion icons. It also assumes that you have a working knowledge of variables and system functions, especially the overlapping function presented in Chapter 4.

The Problem

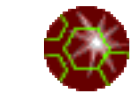
You are a high school history teacher and your class is studying World War II. Instead of giving your class loads of facts to remember, you want them to generate (with your help) the information they think is worth knowing. One strategy is to have them generate their own questions, but their initial reaction to the activity was less than enthusiastic. Writing good questions is hard work they discover. Why bother they say? You've decided to try the idea of game construction as a context for their work. While doing some brainstorming, you discover the students' interest in computer games and mazes. It would be great if there was some sort of maze game template for the computer that the students could use to embed their questions. Maybe Authorware can help.

Introduction

In the previous chapter you learned about an interesting graphics function — "overlapping." However, we saw that this function was not well suited to the rendezvous game in which the player flies their space shuttle to the space station. The project in this chapter, in contrast, is very well suited to this function. We will again create a graphical object under the direct control of the player. However, the goal in this game is to make your way through a simple maze as fast as you can. Of course, the program must prevent players from cheating — they must not be allowed to take shortcuts through the maze! Along the way, players will have to answer questions to get through various gates that block the way. The overlapping function will be used to accomplish all of this. Part of the appeal of the approach we will take is that the maze game we construct will become a sort of template. It will be easy to modify and adapt it for other maze games. The maze itself can be redesigned in infinite ways.

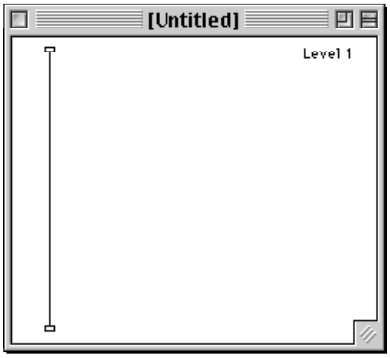
Getting started

The first step is to launch the Authorware application.



☐ **Start Authorware by double-clicking on the Authorware application.**

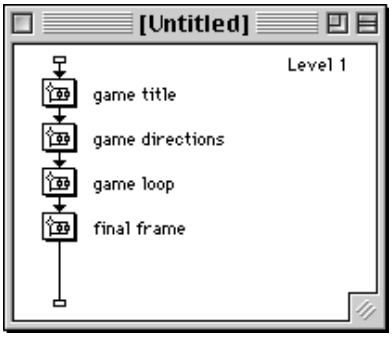
A new file opens with a Level 1 flowchart with the name "untitled":



Setting up the game structure

Just like last chapter, we will start by using Map icons to organize the overall structure of the game's parts.

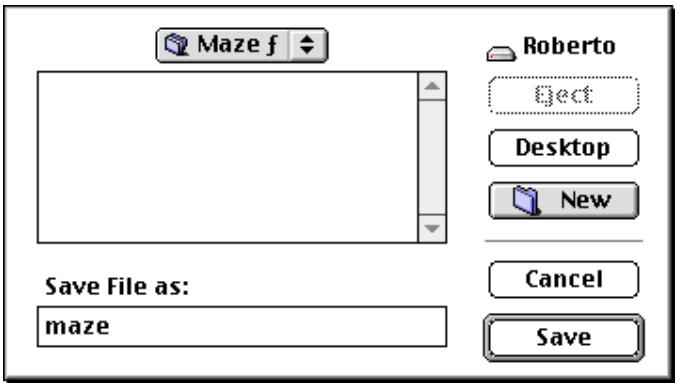
☐ **Drag four Map icons to the flow line and title them as per the following:**



Let's go ahead and save the file.

☐ **Select "Save as..." from the File menu.**

☐ **Save your file with the name "maze" to the folder of your choice.**



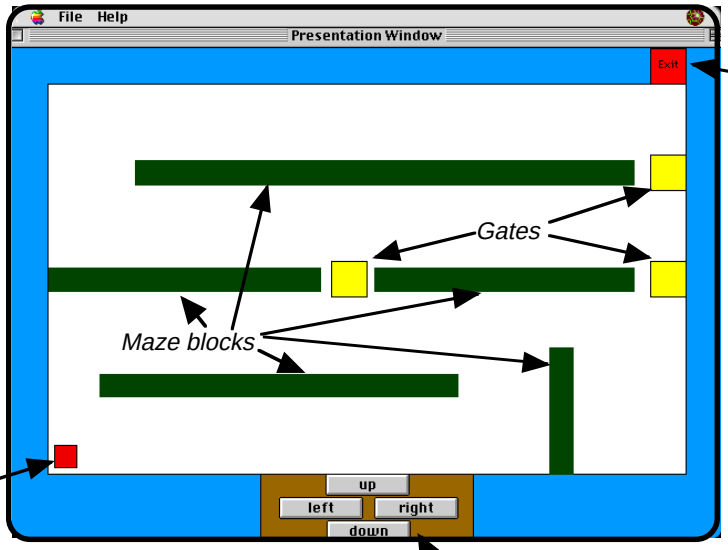
From here on, remember to save your file periodically using the "Save" command from the "File" menu.

Note again that Authorware will automatically add the extension ".a4p" to your file name.

As in the previous chapters, we are not going to take the time to step through the design of a suitable title and directions. Instead, we will start immediately with the game construction.

Let's start by getting familiar with how the game is supposed to be played from the player's point of view.

Here is a snapshot of the game's main screen:



In the working prototype, I colored the player's game piece and maze exit red, the maze blocks green, and the gates yellow. I also painted the background blue and the player control area brown. These colors provide nice contrast.

The goal of the game is for the player to move their game piece through the maze to the maze's exit as fast as they can (the computer's clock will be running throughout the game). The player maneuvers their piece through the maze via the controls at the bottom center of the screen. If the player bumps into one of the maze blocks, the player is "bounced" back. If the player lands on one of the maze gates, a question is presented. If the player answers correctly, the gate is erased and the player moves on. If answered incorrectly, the player's game piece is returned to the starting position (and valuable time is thereby lost).

Of course, we could also penalize the player in some way if they touch a maze block.

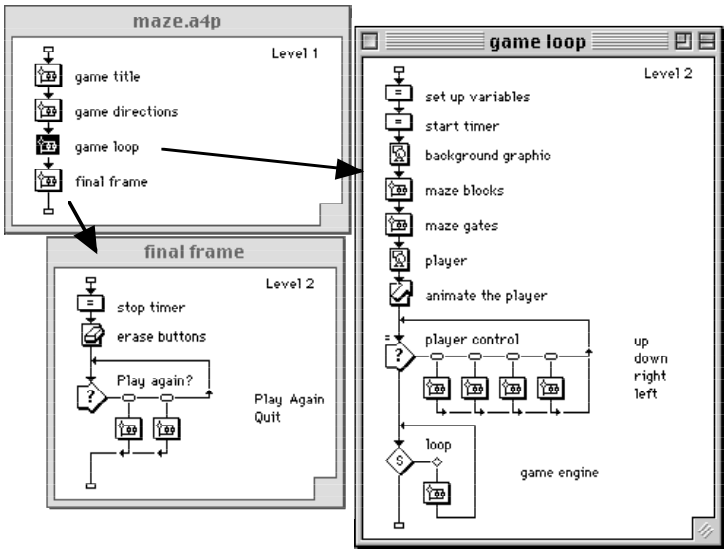
It's important to note that the maze configuration that you see above is just one of hundreds that could be built with these objects. Any of the objects could be moved and stretched anyway you like (at least while the file is unpackaged). While the game is running Authorware simply checks whether or not the player's game piece overlaps any of the other game objects and, if so, triggers the appropriate action. There's no need for any complex mathematical model. What's also important to note at this point is that although we will build our maze game with five maze blocks, three gates, and one exit, the structure and logic of our maze easily permits us to have as many maze blocks, gates, and exits as you wish with very little extra work.

During our game construction, we will keep all our objects rectangular so as to more easily troubleshoot the program (remember that the overlapping function checks the rectangular background of the two objects). Only at the very end will we replace the game piece and gates with more appropriate graphics.

Review chapter 4 if you're confused about this.

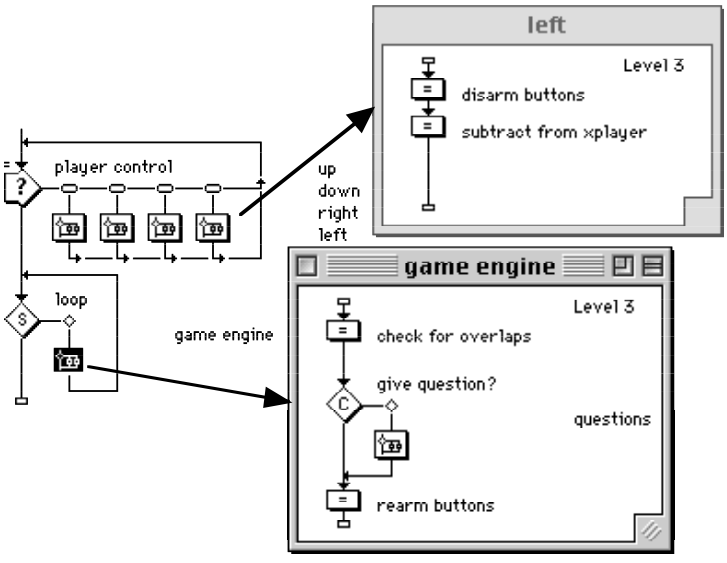
OK, now let's take a look at the critical parts of the completed game from the programmer's point of view.

Here is a graphical overview of the finished game:



If you have successfully completed the other chapters, you should quickly see that we are using the same basic structure here. One difference includes the two Calculation icons "start timer" and "stop timer" which control the timekeeping features of the game. Also note the two Map icons which contain the maze blocks and maze gates.

Here's a closer look at one of the player control buttons (the others use the same structure) and the game engine:



The "left" button includes two Calculation icons. As we will see, it will be necessary to "disarm" or "deactivate" the con-

trol buttons (otherwise the player will be able to cheat!) The second Calculation icon changes a variable used to control the animation of the game piece in a way very similar to that shown in the last chapter (although the piece will only move one space at a time and then stop).

The game engine checks to see if the player's game piece is overlapping any other game object. If so, then the appropriate action is triggered. For example, we need to program Authorware to present a question only when the player lands on a maze gate (that's the purpose of the "give question?" Decision icon). Therefore, it will be necessary to set up a variable that "tells" Authorware if the player's game piece is overlapping one of the gates. Likewise, we will need to set up several other variables that "trigger" appropriate actions in the game.

Here is a brief overview of the game's variables (in the order in which we will use them):
XPLAYER - contains the horizontal location of the player in the animation area.
YPLAYER - contains the vertical location of the player in the animation area.
XPLAYEROLD - stores the player's most recent horizontal position (used when "bouncing" player off of maze block).
YPLAYEROLD - stores the player's most recent vertical position (used when "bouncing" player off of maze block).
GAMEOVER - determines if the game is over: true - yes; false - no.
GATE - keeps track of which gate the player's game piece is currently overlapping.
GIVEQUESTION - determines whether or not to give a question at the end of each and every loop: true -- give question; false -- do not.
PATH - a variable used to determine the calculated path in a Decision icon in each question; value depends on whether the player gets the question right or wrong.
STARTTIM - the time, in seconds, that the game began (based on the computer's system clock).
ENDTIME - the time, in seconds, that the game ended (based on the system clock).
SCORE - the player's score for the round; equal to the number of seconds elapsed since the game began (i.e. STARTTIM minus ENDTIME).
BUTTONARMED - this "arms" or "disarms" the player control buttons; true - armed; false - disarmed.

***Note**: the name "starttime" cannot be used because it is already the name of a system variable; therefore it is reserved by Authorware. We must use another name for our variable.*

On the next page is a complete printout of the contents of the Calculation icon "check for overlaps." Try not to be intimidated or turned off by this. Instead, skim it over now to try to see the patterns. The main reason for providing it here is as a resource for completing the chapter.

Authorware 4 programming in the Calculation icon "check for overlaps":

```
--reset variables:
givequestion:=FALSE
```

This line resets the variable that instructs Authorware as to whether or not to give the player a question. The variable is set to "false," meaning "don't give a question." This variable stays false unless they land on a maze gate.

These check to see if the player's game piece is overlapping one of five blocks which form to make the maze. If they are, Authorware is programmed to snap (or "bounce") the player back to the previous spot.

Notice the pattern in each of these five groups of code. The only difference between them is the title of the Display icon referring to the block number.

```
--check to see if player is overlapping a block:
if Overlapping(IconID@"player", IconID@"block1") then
xplayer:=xplayerold
yplayer:=yplayerold
end if

if Overlapping(IconID@"player", IconID@"block2") then
xplayer:=xplayerold
yplayer:=yplayerold
end if

if Overlapping(IconID@"player", IconID@"block3") then
xplayer:=xplayerold
yplayer:=yplayerold
end if

if Overlapping(IconID@"player", IconID@"block4") then
xplayer:=xplayerold
yplayer:=yplayerold
end if

if Overlapping(IconID@"player", IconID@"block5") then
xplayer:=xplayerold
yplayer:=yplayerold
end if
```

These check to see if the player's game piece is overlapping one of three "maze gates." If they are, variables are set that will soon instruct Authorware to give a question plus erase the gate if the answer is correct.

```
--check to see if overlapping a gate:
if Overlapping(IconID@"player", IconID@"gate1") then
gate:=1
givequestion:=TRUE
end if

if Overlapping(IconID@"player", IconID@"gate2") then
gate:=2
givequestion:=TRUE
end if

if Overlapping(IconID@"player", IconID@"gate3") then
gate:=3
givequestion:=TRUE
end if
```

Again, notice how the code repeats itself for each of the three gates, the only difference being the title of the gate's Display icon.

This checks if the player's object is overlapping a display at the maze exit. If they are, a variable is set that will soon instruct Authorware to end the game.

```
--check to see if overlapping maze exit:
if Overlapping(IconID@"player", IconID@"maze exit") then
gameover:=TRUE
```

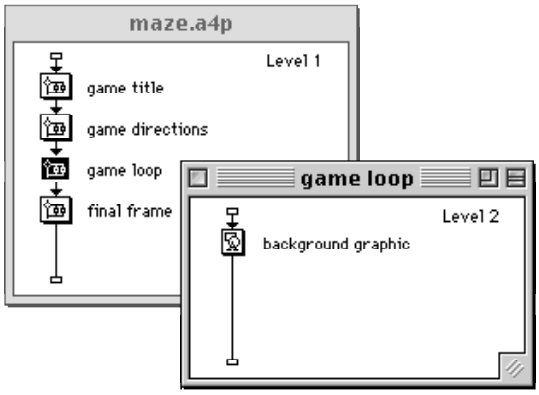
```
xplayerold:=xplayer
yplayerold:=yplayer
```

These two lines tell Authorware to "remember" the current position of the player's game piece. If the player tries to cross a maze block on the next move, Authorware uses these variables to snap the player's object back to this "old" position.

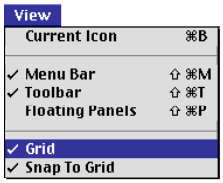
Setting up the game background and maze objects

Constructing this game will be made much easier if we spend a little extra time at the start to decide the basic shape and size of the game board and game objects.

☐ **Double-click to open the "game loop" Map icon and drag a Display icon to the Level 2 flow line; title it "background graphic."**



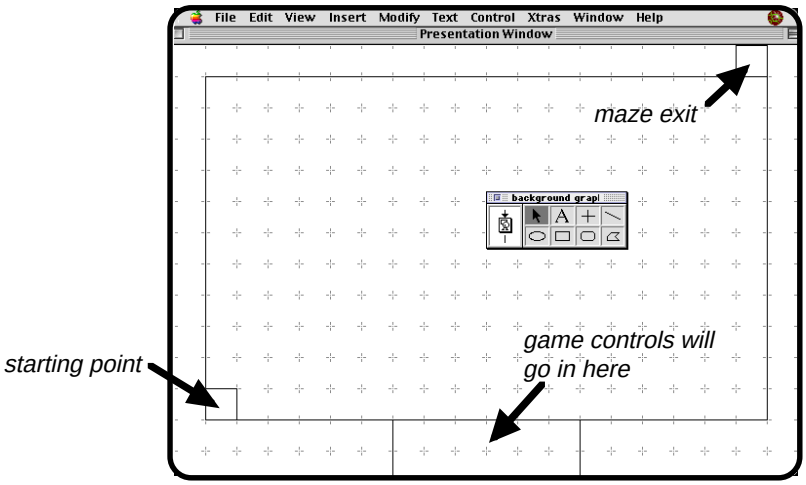
☐ **Double-click to open this Display icon.**



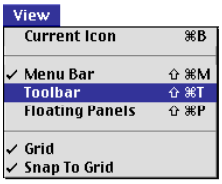
☐ **Turn on "Grid" and "Snap to Grid."**

These two features will be a big help in constructing the background for the game board *and the various game objects*. Think of the gameboard as a 2-dimensional grid on which the player will roam around.

☐ **Construct a game board background similar to the following:**



You might find it useful to turn off the display of the tool bar:



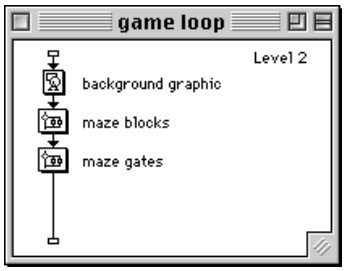
As you can see, we want a grid system with 18 squares across and 12 squares up and down (if you go from the starting position square to the maze exit square).

Again, you can go back later to add color and other graphic elements to your background. For now, this grid will be sufficient to guide the construction of the other game objects.

☐ **Click in the close box in the tool box window to go back to the flow line.**

Next, let's set up two Map icons. One will hold the 5 maze blocks and the other will hold the 3 maze gates and the maze exit.

☐ **Drag two Map icons to the flow line and title them "maze blocks" and "maze gates":**

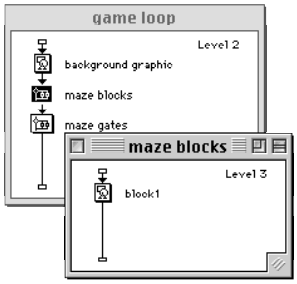


Let's start with building the five maze blocks. In order for the overlapping function to work, each game block must be in its own separate Display icon because the overlapping function can only check an entire Display icon, not individual elements within a Display icon. The strategy here is to construct the first block the way we want it, then we can copy and paste this four times to make the other blocks.

Constructing the maze blocks

☐ **Double-click to open the Map icon "maze blocks."**

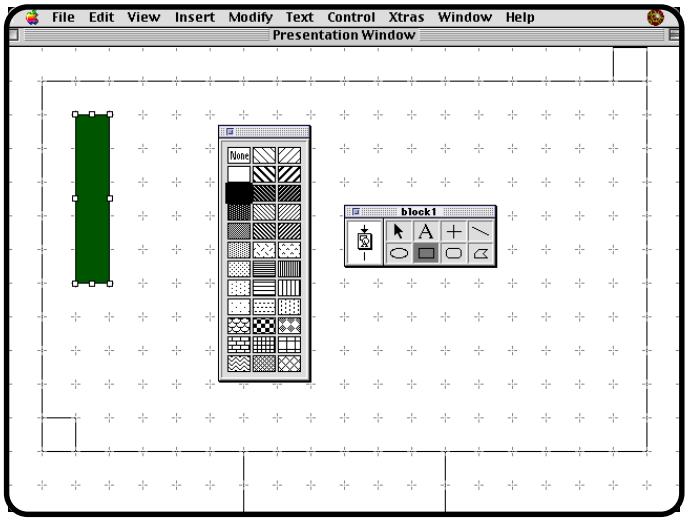
☐ **Drag a Display icon to the Level 3 flow line and title it "block1."**



As we construct this first block, it will be useful to have the background graphic also appear. Let's just choose to run the file. This way, the background graphic will appear first (it comes first on the flow line) followed by the Display icon "block1." Since "block1" is yet undefined, Authorware will conveniently pause the execution of the file and open this icon for us automatically.

☐ **Choose "Restart" under the "Control" menu.**

☐ **Construct a small "tall and skinny" rectangle toward the left hand side of the screen:**



You are also encouraged to fill this rectangle in with a color of your choice (I chose green).

☐ **When finished, click on the close box in the tool box.**

Since we were "running" the file at the time, this merely resumes the execution of the file. We want to go back to the flow line.

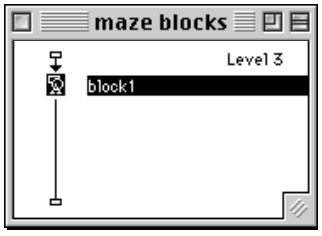
☐ **Press "Command/Control-J" to jump back to the flow line.**

Now we can use the "copy and paste" feature to quickly duplicate this display.

Platform Alert!

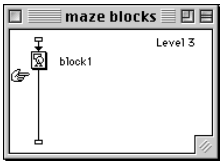
As mentioned in chapter 1, Macintosh users use the Command key and Windows users use the Control key for all keyboard shortcuts. (No further reminders will be given in this chapter.)

☐ **Click once on the Display icon "block1."**



☐ **Choose "Copy" from the "Edit" menu.**

This copies the Display icon (and its contents) to the clipboard.



☐ **Click once on the flow line just below the Display icon "block1" so that the paste hand appears.**

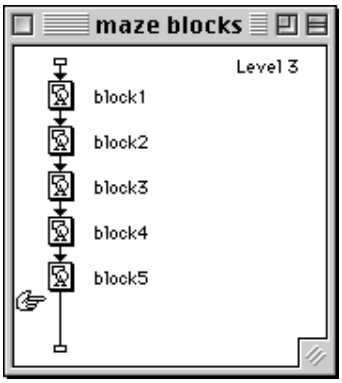
☐ **Choose "Paste" from the "Edit" menu.**

This pastes an exact duplicate of "block1." We want a total of 5 blocks, so we do this three more times.

☐ **Paste three more copies of this Display icon so that a total of five displays with the title "block1" appear on the Level 3 flow line.**

☐ **Rename these as "block2," "block3," etc.**

***Tip**: If you press return when an icon is selected at the flow line, Authorware selects the very next icon and highlights the title — you can just begin typing to rename.*



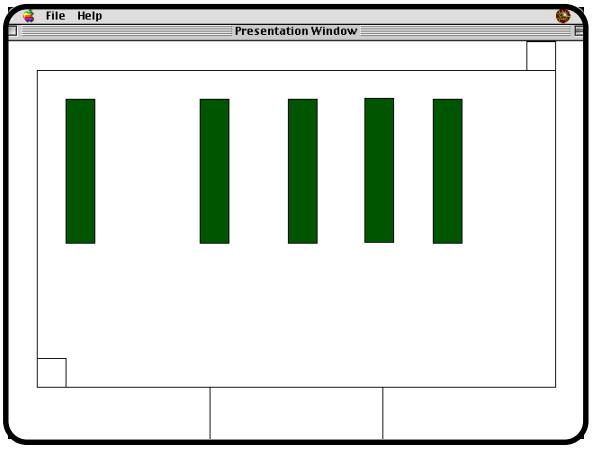
It is *very* important that each of these have distinct names for the overlapping function to work properly. Obviously, the simpler the names, the easier it will be for us to remember these later on.

Now, let's run the file in order to "scatter" these blocks around the game board.

☐ **Restart the file.**

Although it probably appears at first that you still only have one maze block on the screen, realize that you really have all 5 blocks lined up one on top of the other. Even though you are running the file, Authorware allows you to "click, hold, and drag" each block around the screen.

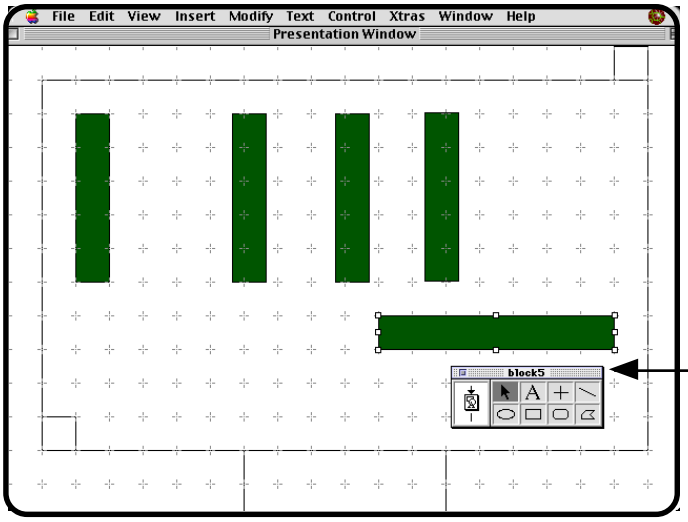
☐ **Drag each of the five blocks to a different spot on the screen:**



You can also double-click on any one of the maze blocks in order to open that particular Display icon. When you do, you can resize the block any way you wish.

☐ **Double-click on the maze block on the far right side.**

☐ **Resize this maze block to a horizontal position:**



Notice that the title of the Display icon is shown here.

Take time to notice that the title in the tool box corresponds to the title of the Display icon that is currently open. This is the only way to tell which display is being edited.

☐ **When finished, click in the close box in the tool box.**

You could continue to double-click on the other blocks in order to resize them as desired. In this way, you can create a large variety of mazes. We'll do more of this later. For

now, let's create the other game objects, starting with the maze gates and maze exit.

☐ **Press Command/Control-J to go back to the flow line.**

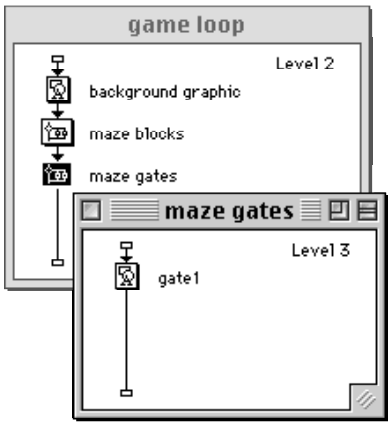
Constructing the maze gates and maze exit

We will use the same basic procedures for creating the maze gates and maze exit — construct one, copy and paste it three times, then rename each icon (the maze exit will have the same shape as a maze gate).

☐ **Close the "maze blocks" Map icon window.**

☐ **Double-click on the "maze gates" Map icon.**

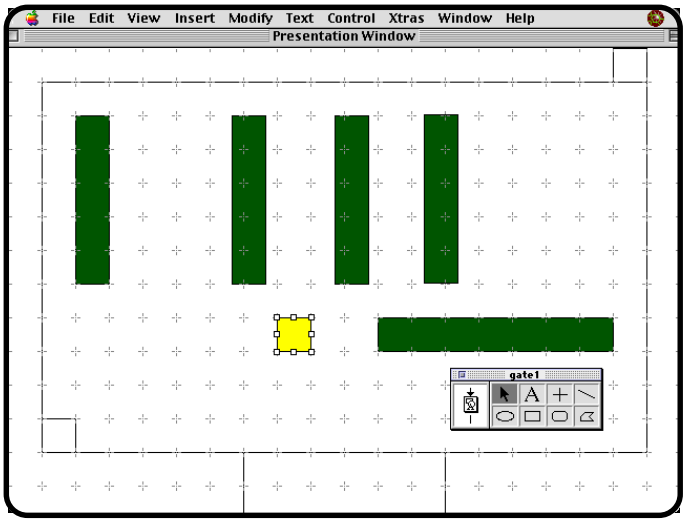
☐ **Drag one Display icon to this Level 3 flow line and title it "gate1."**



Again, we'll use the same trick of just running the file to have Authorware display all of the previous displays and have it automatically open up this Display icon for us to define.

☐ **Restart the file.**

☐ **Construct a small square exactly one "grid block" large somewhere in the middle of the screen:**



Fill this in with a color of your choice (I picked yellow).

☐ **When finished, click in the close box in the tool box.**

☐ **Press Command/Control-J to go back to the flow line.**

Now we copy and paste this display three times.

☐ **Click once on the "gate1" Display icon.**

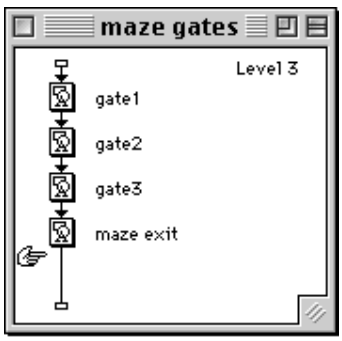
This selects the icon for copying.

☐ **Choose "Copy" from the "Edit" menu.**

☐ **Click once just below the "gate1" Display icon so that the paste hand appears.**

☐ **Paste this icon three times.**

☐ **Rename the displays "gate2," "gate3," and "maze exit":**



☐ **Restart the file.**

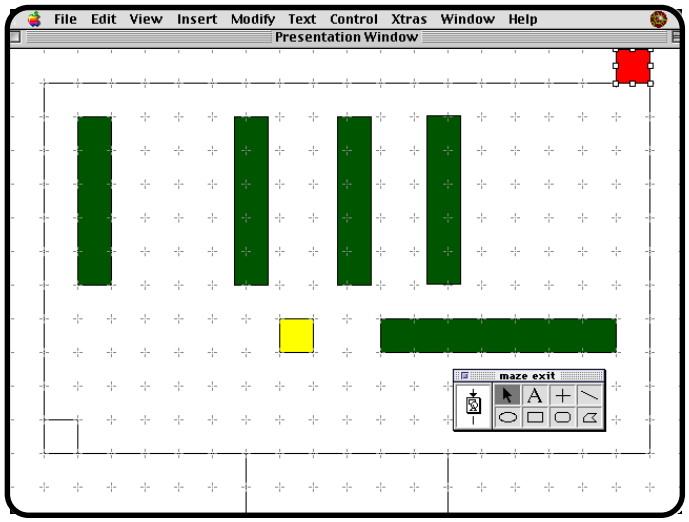
Again, although there appears to be only one maze gate on the screen, the three maze gates plus the maze exit are just lined up one on top of the other. In fact, the maze exit is the one on top (since it was the last icon displayed). Let's put this exit where it belongs up in the top right-hand corner of the gameboard.

☐ **Double-click on the "stack" of maze gates.**

By opening this icon, the grid lines will also appear (assuming they were turned while you were editing the previous display).

Be sure that the title "maze exit" appears in the tool box.

☐ **Drag the maze exit to the top right square.**

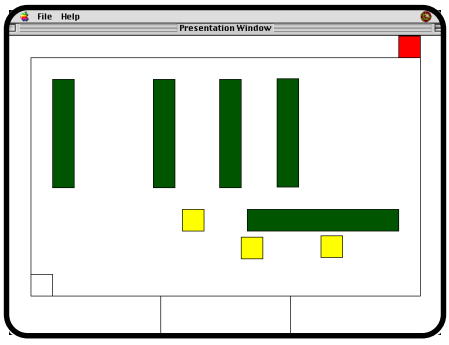


I also recommend that you go ahead and change the color of this maze exit (I chose red).

☐ **Click in the close box in the tool box.**

Authorware resumes "running" the file. If no icons are open, we can freely move any graphic display. So, let's "scatter" the maze gates. I suggest dragging the gates to the right, so that gates 1, 2, and 3 will line up left to right on the screen.

☐ **Drag the maze gates to different spots on the screen.**



Remember, you can double-click on any maze gate to determine which one it is by looking at the title in the tool box.

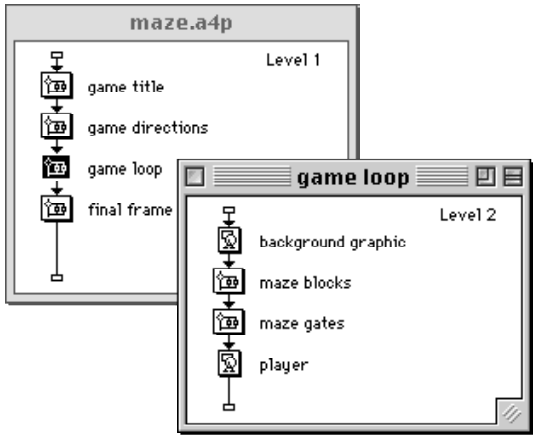
That's fine for now. Let's jump back to the flow line and begin working on the player's game piece and the animation that controls it.

☐ **Press Command/Control-J to go back to the flow line.**

Constructing the player's game piece

We again must take care as we build the player's game piece to construct it with the grid system in mind and how the overlapping function works. First, we will construct a game piece that is exactly one game square in size. Next, we will set up the animation that controls it. Finally, we will edit the game piece graphic to "shrink" it down just a little to make sure it does not overlap at inappropriate times.

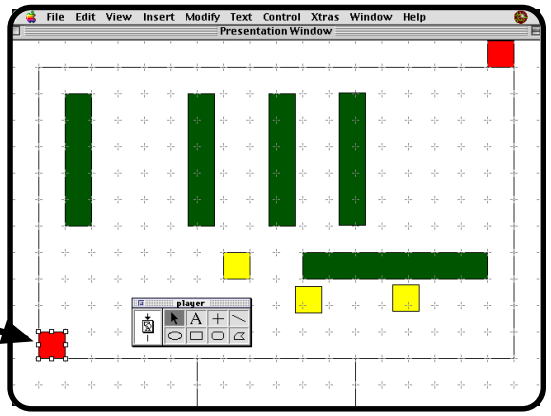
☐ **Drag a Display icon to the Level 2 flow line inside the Map icon "game loop" and title it "player."**



As before, we will just run the file and let Authorware display all of the previous graphics as well as automatically open up this as yet undefined icon.

☐ **Restart the file.**

☐ **Construct a simple square graphic that is exactly one game square in size and put it in the starting point of the gameboard.**



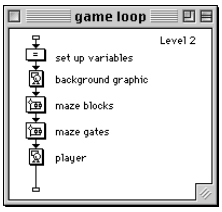
Player's game piece placed at the game's starting point.

☐ **When finished, click in the close box in the tool box.**

☐ **Press Command/Control-J to go back to the flow line.**

Setting up the animation for the player's game piece

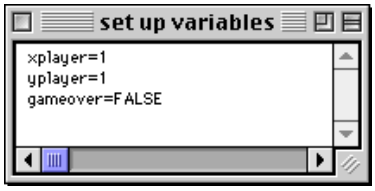
Let's start by setting up the variables that will control the animation.



☐ **Drag a Calculation icon to the top of the Level 2 flow line and title it "set up variables."**

☐ **Double-click on this icon to open it.**

☐ **Type the following in the Calculation icon's text window:**



The first two lines set up two variables that will control the horizontal ("xplayer") and vertical ("yplayer") motion of the player's game piece (similar to that shown in previous chapters). We want the piece to be at the starting position, so we set these variables to 1.

While we are here, we might as well set up the variable that "tells" Authorware when the game is over. This is the purpose of the third line in which the logical variable "gameover"

is set to FALSE. Think of it as answering the question "True or false, is the game over?" This variable will control when the game loop should end (more about that later).

That's all for now.

Logical variables can only store the value of true or false.

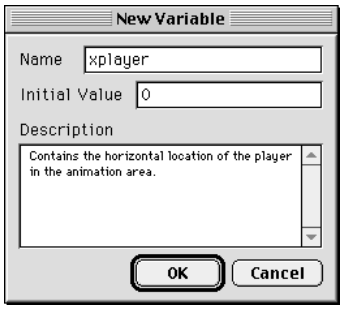
Logical variables are one of the three types of variables that Authorware uses, the other two being numeric and character.

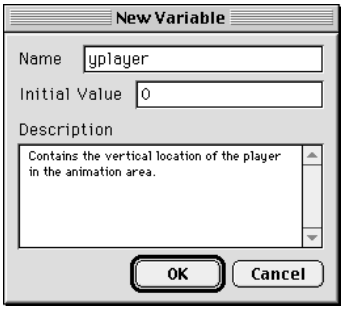
☐ **When finished, click on the close box in the Calculation icon's window.**

☐ **Click the "Yes" button to save the changes to this icon.**

All three are new variables, so new variable dialog boxes automatically open for each.

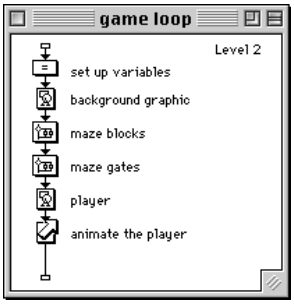
☐ **Type in the following initial values and descriptions for each of the following three new variables, and click "OK" after each one.**





We are now ready to construct the animation sequence using the Motion icon.

☐ **Drag a Motion icon to the Level 2 flow line and title it "animate the player."**



We can once again just run the file and let Authorware open this icon automatically (with all of the graphics appearing first on the screen).

☐ **Restart the file.**

All the game graphics should appear before the Motion icon's dialog box pops up. We will need to change the motion type.

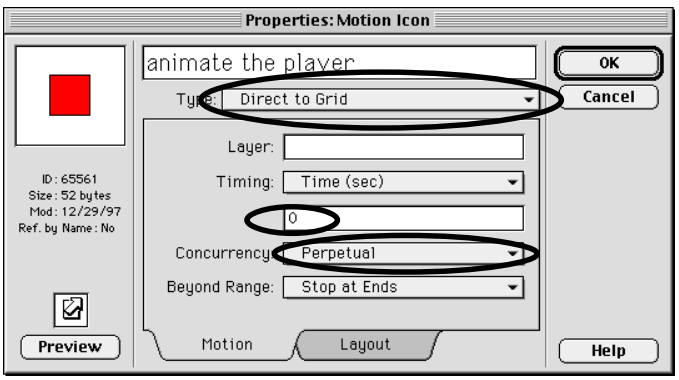
☐ **Click and hold on the words "Direct to Point" to activate the pop-up menu, then select "Direct to Grid".**

While viewing the Motion options, let's go ahead and change the timing to zero (0) seconds and the concurrency to Perpetual so that the animation will update itself instantaneously during the game.

☐ **Enter a zero ("0") in the text box below timing.**

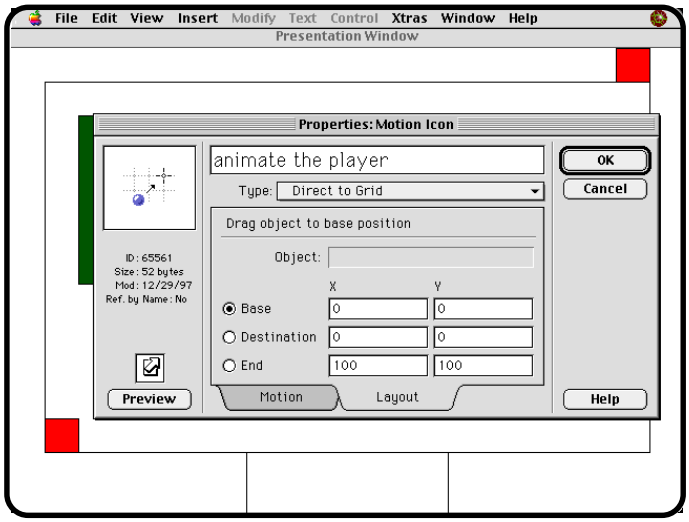
☐ **Change the Concurrency from "Wait Until Done" to "Perpetual."**

When done, this Motion icon's dialog box should match the following:



☐ **View the "Layout" options by clicking once on the Layout tab.**

Your screen should resemble the following:



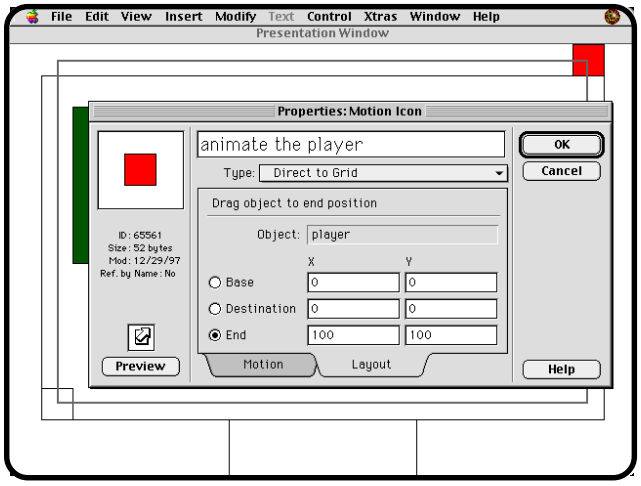
The directions in this dialog box say to "Drag object to base position." Well, if you constructed the player's game piece precisely as above, it already *is* in the starting position. So, all we need to do is click once on it — this tells Authorware which graphic display is to be animated plus tells it where the base position is located.

☐ **Click once on the player's game piece, but be careful not to move it.**

Now let's tell Authorware where the "end position" is located (directly on top of the maze exit).

☐ **Click on the radio button beside "End," then drag the player's game piece to the "end position" directly on top of the maze exit in the top right corner of the gameboard.**

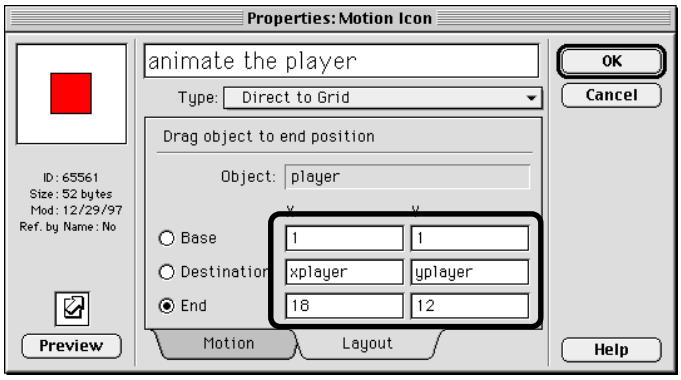
You should see the “fuzzy rectangle” defining the animation area:



Let’s make the final modifications to the Motion icon.

☐ **Make the following modifications to the Motion dialog box:**

1) **type “xplayer” and “yplayer” in the two text windows beside Destination;**
2) **Enter “1” for both the base positions;**
3) **Enter “18” for the X end position;**
4) **Enter “12” for the Y end position.**



The first modification tells Authorware to use the variables we set up earlier to control this animation. The other modifications define the animation area’s dimensions according to the grid of our gameboard (there is no “0” game square).

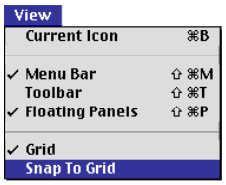
☐ **When finished, click “OK.”**

☐ **Press Command/Control-J to go back to the flow line.**

Resizing the player's game piece

Now that the animation has been defined successfully, the final step is to resize the game piece so that it is slightly smaller than one game square. We do this because even if only the *edges* of two graphical objects touch, Authorware perceives this as overlapping.

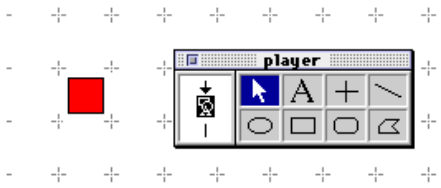
☐ **Double-click on the Display icon "player."**



☐ **Turn off "Snap to grid."**

If the grid is on, it will prevent us from changing the size a "teeny" bit.

☐ **Resize the game piece so that it is safely a few pixels smaller than the game square (as defined by the grid):**



☐ **When finished, click in the close box in the tool box to go back to the flow line.**

Setting up the player's game controls

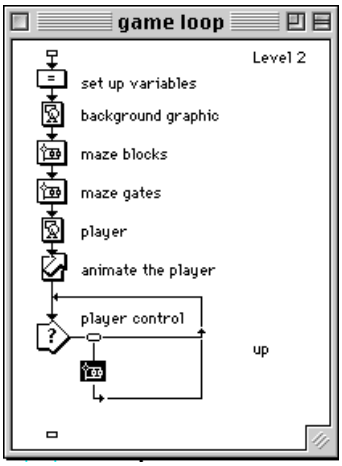
We need to set up four perpetual pushbuttons to give the player control over the game piece. The procedures to create these four buttons are virtually identical to those in the previous chapter. The only difference is that instead of following Newton's laws of motion, we just want the game piece to move one square at a time and then stop.

☐ **Drag an Interaction icon to the Level 2 flow line just below the Motion icon and title it "player control."**

☐ **Drag a Map icon to a point just right of the interaction.**

☐ **Make sure it is a "Button" response type, then click "OK" in the response dialog box.**

☐ **Title this Map icon "up."**

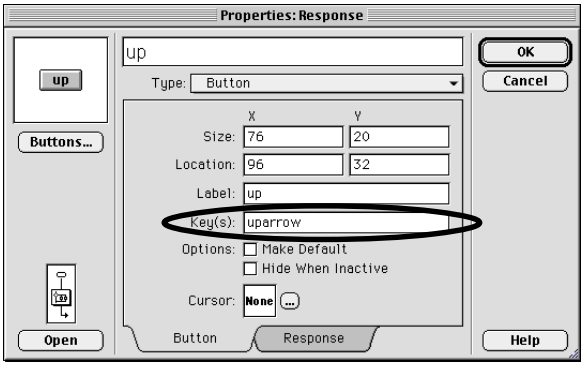


The first thing we will do is construct this button so that the player could also press the up arrow on the keyboard instead of using the mouse when playing — keyboard controls are often preferred by game players.

☐ **Double-click on the baby pushbutton just above the branch leading into the Map icon "up."**

This opens the dialog box for this branch (with the Button options mostly likely displayed).

☐ **Type "uparrow" in the text window beside "Key(s)."** (Be sure to type "uparrow" as one word without a space.)



Be sure you are viewing the Button options.

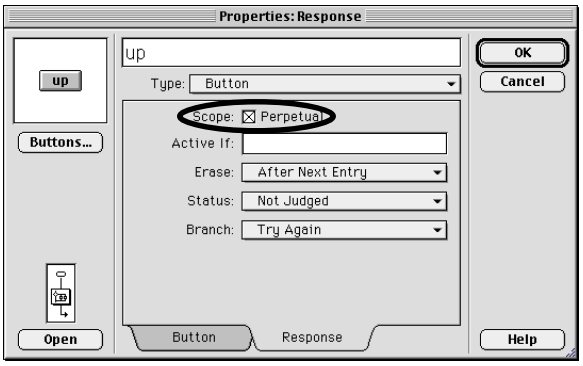
This simply means that pressing the optional key triggers this button just as if the button on the screen was clicked. There is a way to identify virtually every character on the keyboard. Some characters are straightforward, such as "U" or "u" to identify the upper or lower case letters. Special keys, such as the arrows, have less obvious names: uparrow, downarrow, rightarrow, and leftarrow.

You can even identify special keystrokes, such as Command-7 or Control-Z by typing "Cmd7" and "Ctrlz."

Next, we need to change this to a perpetual pushbutton so that the button is always active.

☐ **View the Response options by clicking once on the Response tab.**

☐ **Click once in the small check box beside "Perpetual."**



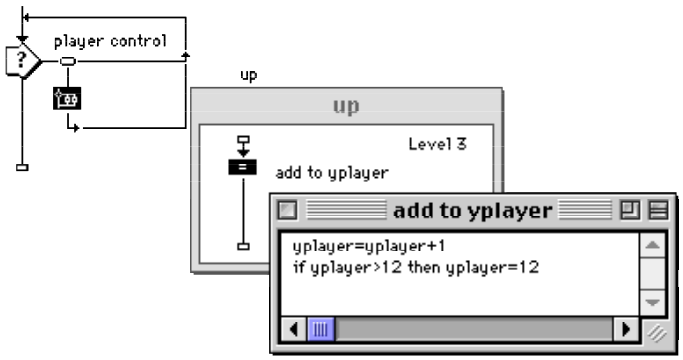
☐ **When finished, click "OK."**

Now let's set up the action that this button will perform by putting a Calculation icon inside its map.

☐ **Double-click on the Map icon "up."**

☐ **Drag a Calculation icon to the Level 3 flow line and title it "add to yplayer."**

☐ **Double-click on this Calculation icon and type the following in the text window:**



The first line tells Authorware to increase "yplayer" by 1. Since the animation area goes from 1 to 12 vertically, the game piece will move up to the next square.

The second line tells Authorware that if the game piece is moved beyond 12 to reset it back to 12. In this way, the variable "yplayer" can never go above 12. (We will use this same logic in the other buttons.)

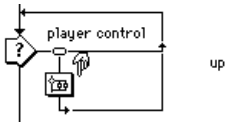
☐ **When finished, click on the close box to close the Calculation icon's window, then click "yes" to save the calculation changes.**

☐ **Close the Map icon "up" (containing the Level 3 flow line).**

As in the previous chapter, we will copy and paste this button to save us some work in constructing the "down" button.

☐ **Click once on the Map icon "up."**

☐ **Choose "Copy" from the "Edit" menu.**



☐ **Click just to the right of the "up" button branch so that the paste hand appears.**

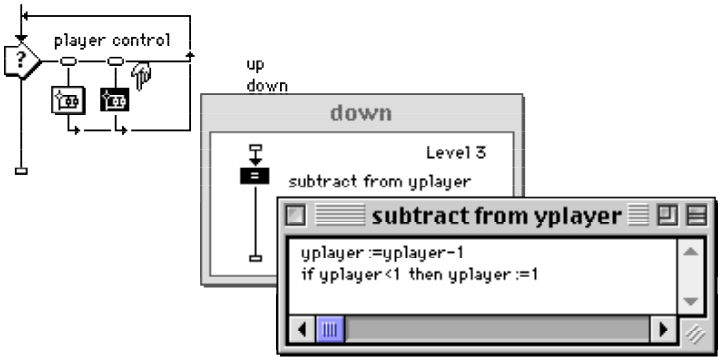
☐ **Choose "Paste" from the "Edit" menu.**

☐ **Change the name of this branch (or Map icon) to "down."**

☐ **Double-click to open the "down" Map icon.**

☐ **Change the name of the Calculation icon to "subtract from yplayer."**

☐ **Double-click to open this Calculation icon and make the following changes:**



In the first line, we change the + (plus sign) to - (minus sign) because when player clicks on this button the game piece needs to move down the gameboard.

The second line makes sure that "yplayer" will never have a value lower than "1."

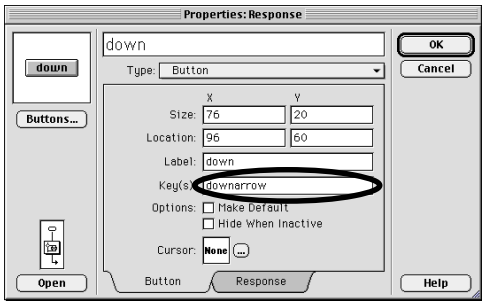
☐ **When finished, click on the close box to close the Calculation icon's window, then click "yes" to save the calculation changes.**

☐ **Close the Map icon "down" (containing the Level 3 flow line).**

We also need to change the optional key for this button from "uparrow" to "downarrow."

☐ **Double-click on the baby pushbutton leading into the "down" branch.**

☐ **Change the optional key to "downarrow"** (no spaces)**.**



Be sure you are viewing the Button options.

☐ **Click "OK."**

Now we set up the "left" and "right" buttons the same way. The difference is that these increase or decrease the variable "xplayer." (We also need to change the "optional key" to "leftarrow" and "rightarrow," respectively.)

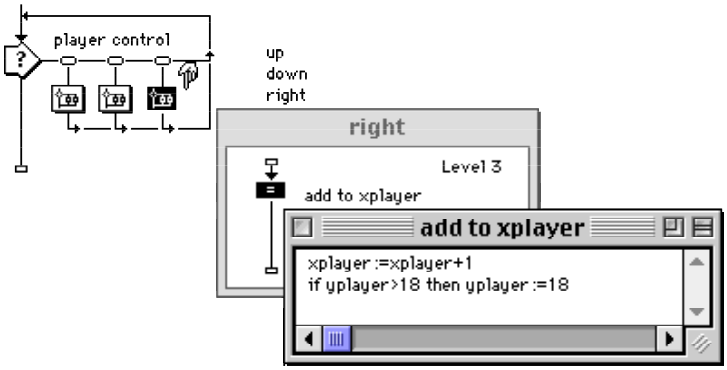
You can either construct these from scratch or copy, paste, and then modify one of the existing branches. Here are abbreviated instructions for doing the latter:

☐ **Copy the "up" branch and paste it just to the right of the "down" branch; title this third branch "right."**

☐ **Double-click on the baby pushbutton leading into the "right" branch and change the optional key to "rightarrow"; click "OK" when finished.**

☐ **Open the "right" Map icon and change the name of the Calculation icon inside to "add to xplayer."**

☐ **Open this Calculation icon and change the variable to “xplayer” as well as the upper range in the if-then statement to 18:**



☐ **When finished, close the Calculation icon window and the “right” Map icon window.**

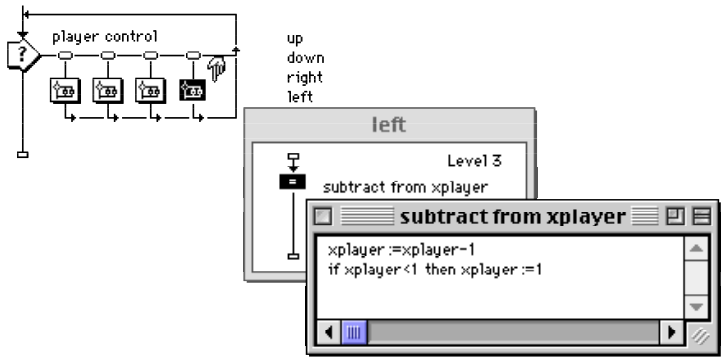
Now let’s do the same to make the fourth (and final) button branch “left.”

☐ **Copy the “down” branch and paste it just to the right of the “right” branch; title this fourth branch “left.”**

☐ **Double-click on the baby pushbutton leading into the “left” branch and change the optional key to “leftarrow”; click “OK” when finished.**

☐ **Open the “left” Map icon and change the name of the Calculation icon inside to “subtract from xplayer.”**

☐ **Open this Calculation icon and change the variable to “xplayer”:**



☐ **When finished, close the Calculation icon window and the “left” Map icon window.**

Testing the interaction and animation

Believe it or not, we can actually run the file at this point even though we have not as yet constructed the loop containing the game engine.

☐ **Restart the file.**

As you click on the buttons, the player's game piece should move accordingly. When you reach the edge of the gameboard, the game piece should just stop. Be sure to test all the buttons at all the boundaries of the gameboard. Also be sure to test pressing the keyboard arrow keys.

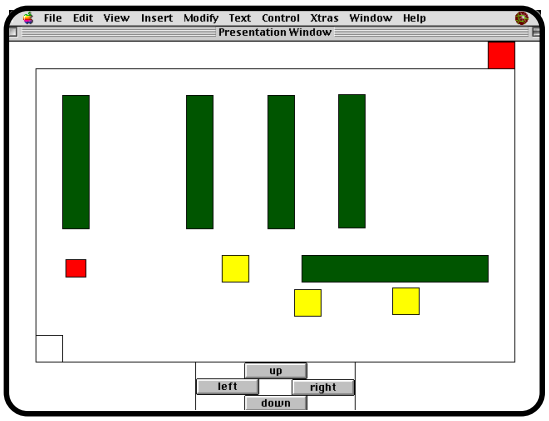
Since we have yet to build in the overlapping function, feel free to move right over the maze blocks and gates.

If the animation does not work properly, you will need to go back and retrace your steps through the procedures.

One thing you probably have already noticed is that the buttons are not conveniently (or meaningfully) placed on the screen. Let's take care of positioning them in the game control area now.

☐ **Press Command/Control-P to pause the file (this assumes you are still running the file).**

☐ **Drag each of the buttons down into the game control area and resize them as necessary:**



☐ **When finished, press Command/Control-P to proceed.**

Keep testing until you are sure that the program is working properly.

☐ **Press Command/Control-J to jump back to the flow line and save your file.**

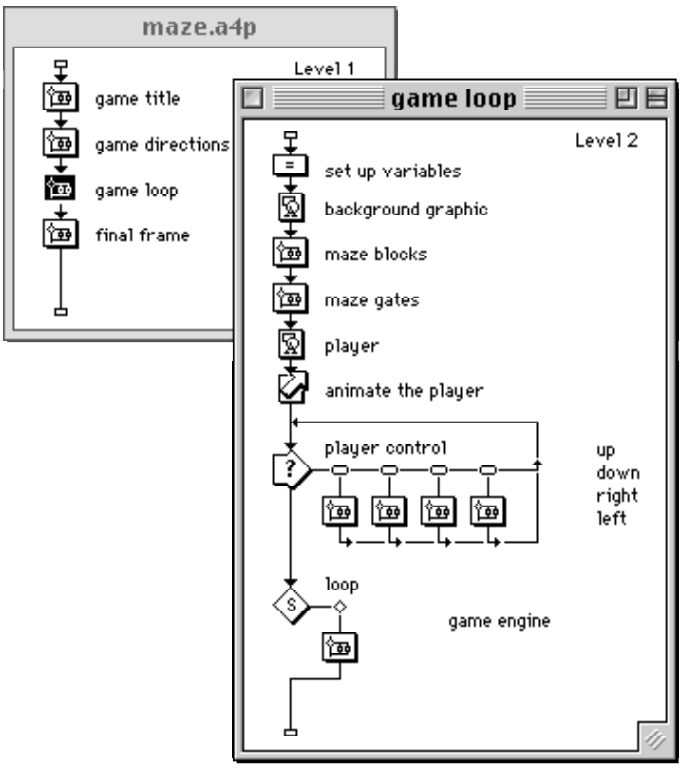
Setting up the game engine loop

The next step is to set up the "game engine" which will continually check to see if the player's game piece is overlapping any maze object. The basic file structure is identical to that used in other chapters — a Decision icon that loops until some condition causes it (and the game) to stop.

☐ **Drag a Decision icon to the bottom of the Level 2 flow line inside of the "game loop" Map icon, and title it "loop."**

☐ **Drag a Map icon just to the right of this Decision icon (to attach it) and title it "game engine."**

Your screen should resemble the following:



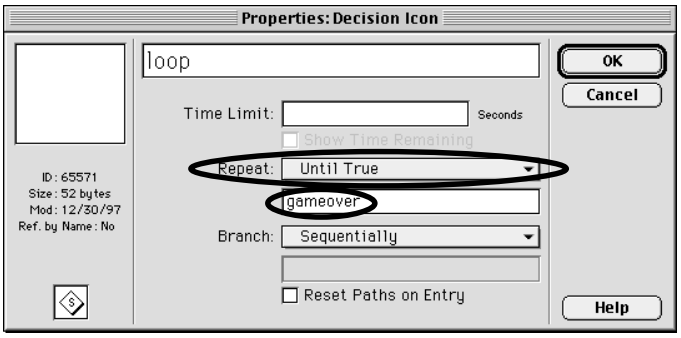
Next, let's modify this Decision icon to loop repeatedly until a certain condition is true, namely, that the game is over.

☐ **Double-click on the Decision icon "loop."**

We already have a variable set up for this job — "gameover." Recall that it is currently set to FALSE. (In a few minutes we will program Authorware to set it to TRUE when the player's game piece overlaps the maze exit.)

☐ **Change the Repeat option from “Don’t Repeat” to “Until True.”**

☐ **Type “gameover” in the text box under “Repeat” “Until True”:**



Therefore, the loop will continue to repeat until the variable “gameover” is changed to TRUE.

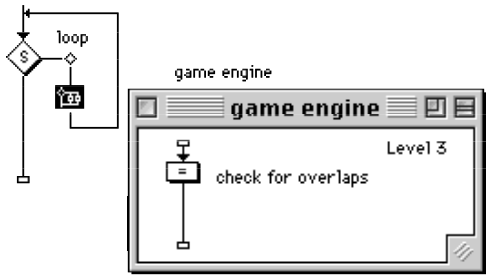
☐ **Click “OK.”**

Now we are ready to construct the first part of the game engine that checks for overlapping between the player’s game piece and the other maze objects.

Programming Authorware to check for overlapping

☐ **Double-click on the Map icon “game engine.”**

☐ **Drag a Calculation icon to the Level 3 flow line and title it “check for overlaps.”**



☐ **Double-click on this Calculation icon to open its text window.**

Let’s start by programming the part that checks if the player’s game piece is overlapping any of the maze blocks. If it is, then we want Authorware to snap or bounce the game piece back.

We will work first only on getting this to work with the maze block titled "block1." Then we can copy, paste, and modify the programming code for the other four blocks.

☐ **Type the following in the Calculation icon's text window:**

Remember that you can also choose to first have Authorware paste this overlapping function in for you and then modify it. Go to "Functions" under "Window"; it is in the graphics category of functions.

***Alert**! Before going on, double-check that you typed these last two lines correctly. The "old" variables go first — it makes a big difference.*

The first line is just a comment (recall that Authorware ignores any line that starts off with two hyphens).

The next four lines are all part of an IF-THEN construction. Here's an explanation. The first tells Authorware that IF the condition (player's game piece is overlapping the first maze block) is true, then do whatever comes next. As you can see, there are two lines that tell Authorware to "swap' the contents of "xplayer" and "yplayer" with that of "xplayerold" and "yplayerold." These "old" variables simply keep track of the player's most recent position *before* they clicked on a button. The line "end if" tells Authorware that the instructions in the IF-THEN group have concluded.

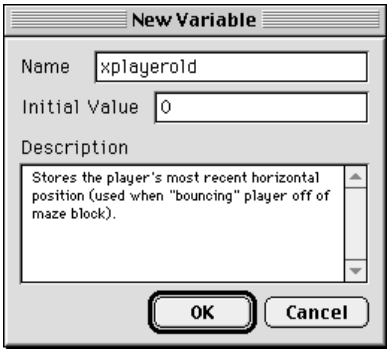
The two lines that come next are what enable Authorware to "know" where the game piece was last. At the end of the game engine, these two lines hold the information of the player's current position and, as you now can see, are used to return the player there if they happen to bump into a maze block.

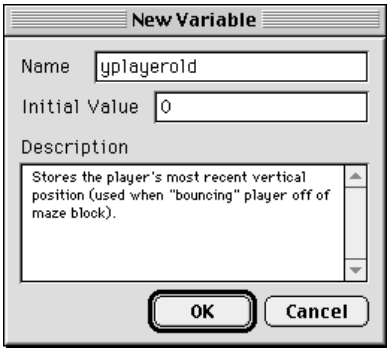
☐ **When finished, click on the close box in the Calculation icon's window.**

☐ **Click the "Yes" button to save the changes to this icon.**

We have two new variables here, so new variable dialog boxes automatically open for each.

☐ **Type in the following initial values and descriptions for each of the following two new variables, and click "OK" after each one.**



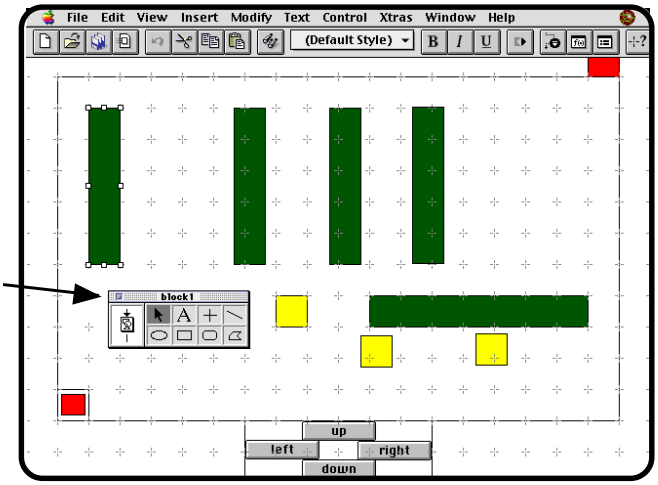


Let's test this. Obviously, it is only supposed to work when the player's game piece overlaps "block1." So, let's be absolutely certain we know which game block is "block1."

☐ **Restart the file.**

☐ **Double-click on the maze block to the far left of the screen and note whether the title "block1" appears in the tool box.**

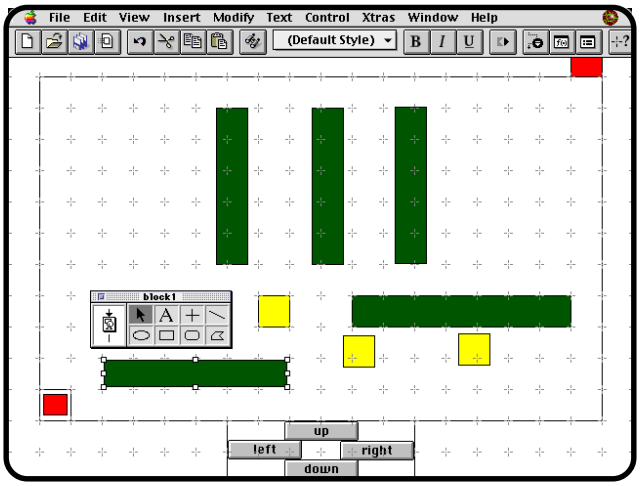
Make sure the title "block1" appears here.



If it doesn't, keep double-clicking on other maze blocks until you find it.

Let's go ahead and move this maze block so that it is close to the game piece. Also make sure you use the grid to position it precisely in one of the grid "channels."

☐ **Move and resize the "block1" maze block to a point similar to the following:**



Turn the grid on or off as necessary. I resized this maze block so that it is a few pixels smaller than the grid area in which I put it.

☐ **When finished, click in the close box in the tool box to resume running the file.**

☐ **Move the game piece toward "block1" by slowly pressing the screen buttons or the keyboard arrows.**

When you try to cross over the maze block, you should see your game piece "bounced" back to the prior spot. (If not, you need to go back and check the programming in the Calculation icon "game engine.")

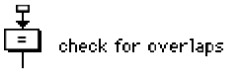
On fast computers, the bounce may be so quick that the square may appear instead to just stand still.

Bug alert! Unfortunately, if you press the keyboard arrows fast enough, you might just move right over the maze block. What's wrong? Well, recall how a perpetual interaction works. Whenever one of these buttons is pressed, the flow is directly taken to the point indicated by the baby pushbuttons. If the button is pressed quickly enough several times in a row, Authorware does not have a chance to execute all of the contents of the Calculation icon "game engine." (In earlier versions of Authorware, this problem occurs even when the screen buttons are pressed rapidly in succession. The problem remains in Authorware 4 only for the optional key presses.) However, the game piece continues to move because the programming of "xplayer" and "yplayer" occur in the button branches, not the game engine. As long as you press arrows slowly, there is no problem. Obviously, this won't do! Later, we will return to fix

this problem by "deactivating" the buttons completely while Authorware completes the instructions in the game engine.

Fortunately, the programming of the other four game blocks is identical except for the name of the block. So, we will copy and paste the programming for block1 four times and make the small modifications for each.

- ☐ **Press Command/Control-J to jump back to the flow line.**



- ☐ **Double-click on the Calculation icon "check for overlaps."**

- ☐ **Highlight all of the text containing the IF-THEN programming.**

- ☐ **Choose "Copy" from the "Edit" menu.**

- ☐ **Click on the empty line below "end if" and choose "Paste" from the "Edit" menu.**

- ☐ **Choose "Paste" three more times.**

- ☐ **Change "block1" to "block2" in the first line of the second IF-THEN group, "block3" in the next group, etc.**

When done, your text window should look like this (I like to put empty lines in between the groups, but they are not necessary):

Yikes! That's a lot of code! Hopefully, you can see how we can quickly reuse and modify the programming logic.

Take one last look at each of the IF-THEN groups to make sure that the maze block title is correct in each.

☐ **When finished, click in the close box in the Calculation icon's window; click "Yes" to save the changes to this icon.**

Time for a test.

☐ **Restart the file.**

☐ **Move the game piece around the screen and test the overlapping of each maze block** (if using the arrow keys, remember to press them slowly)**.**

☐ **Press Command/Control-J to go back to the flow line when you are satisfied the game is working properly.**

Realize that you can double-click on any maze block while running the file to move or resize it, thus making the maze anyway you wish. The game engine doesn't care where the blocks are, only if the game piece is overlapping any one of them.

Maze gates

As you know, the educational content of this game comes from questions that are presented when the player lands on a maze gate. Therefore, we need to do some programming in the game engine in order to trigger the questions at the appropriate time. Then, we are going to construct a simple generic question that can be used as a template for the many questions that ultimately will be included in the game.

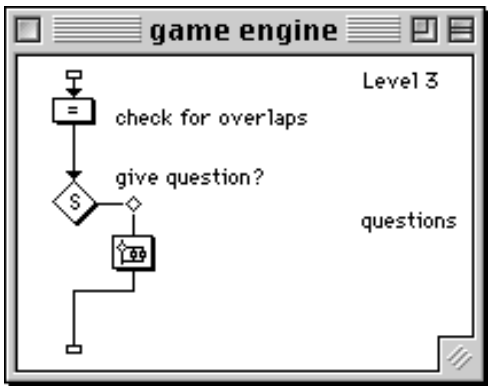
Branching to a question at just the right time

It must be emphasized that a question is not presented until the player lands on a maze gate. Therefore, we will set up the questions as a branch to a Decision icon. The branch is taken only when this condition is true.

- ☐ **Drag a Decision icon to the bottom of the Level 3 flow line inside of the "game engine" Map icon, and title it "give question?".**

- ☐ **Drag a Map icon just to the right of this Decision icon (to attach it) and title it "questions."**

When done, your screen should resemble the following:

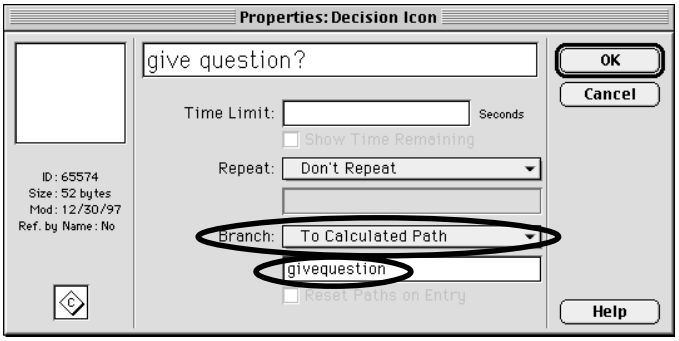


Next, we need to change this Decision icon to branch to "questions" only when the player's game piece overlaps a maze gate. One way to do this is to set up a logical (true/false) variable that is set to TRUE only when this overlapping condition is met. We can then use this variable in this Decision icon to control the branching — branch to questions when true, bypass questions when false. The logical variable we will set up will be called "givequestion."

- ☐ **Double-click on the Decision icon "give question?"**

☐ **Change the Branch option from "Sequentially" to "To Calculated Path."**

☐ **Type the variable name "givequestion" in the text box below "Branch: To Calculated Path":**



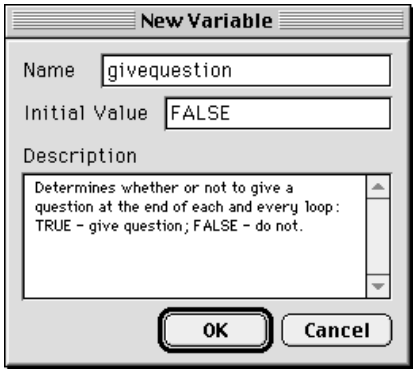
Let's take a moment and explain what this will do. First, the branch option "To Calculated Path" simply tells Authorware which branch to take. Each branch is numbered starting with 1 with 0 meaning "don't take any branch." As you know, we only have one branch attached to this Decision icon called "questions." So, we want Authorware to take this branch (branch number 1) when the logical variable "givequestion" is TRUE. Fortunately, Authorware interprets TRUE as 1 and FALSE as 0. Therefore, the branch will neatly be controlled by this variable either taking the branch (and thereby giving the player a question) when it is TRUE or bypassing the branch when it is FALSE. We will do the programming that sets this variable to TRUE or FALSE in the Calculation icon "check for overlaps" in a few minutes. First, let's finish this step.

Why? It just does.

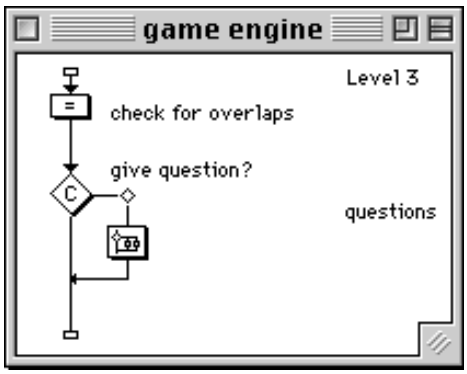
☐ **Click "OK."**

This closes the Decision icon's dialog box. Authorware notices the new variable we are using, so it automatically opens up a new variable dialog box for us to complete.

☐ **Type in the following initial value and description for the logical variable "givequestion," then click "OK":**



The flow line will have changed to reflect the change to the Decision icon's branching:



The "C" inside the diamond stands for "Calculated path"

Now, let's add the programming that triggers the question when the player lands on a maze gate.

☐ **Double-click on the Calculation icon "check for overlaps."**

☐ **Add the following two lines at the very top:**

The first line is again just a comment to ourselves. The second line sets the variable "givequestion" to FALSE at the start of *every* game loop. This will ensure that no question will accidentally be triggered at an inappropriate time.

Now, let's add the programming that tells Authorware to check if the player's game piece is overlapping any of the gates. Frankly, it doesn't matter if we put this code before or after the programming for the blocks (note that on page * it is shown after).

☐ **Add the following lines:**

```
--check to see if overlapping a gate:
if Overlapping(IconID@"player", IconID@"gate1") then
gate=1
givequestion=TRUE
end if

if Overlapping(IconID@"player", IconID@"gate2") then
gate=2
givequestion=TRUE
end if

if Overlapping(IconID@"player", IconID@"gate3") then
gate=3
givequestion=TRUE
end if
```

***Tip**: Remember that you can use the "copy, paste, then modify" trick used earlier.*

These lines tell Authorware that if the player's object is overlapping gates 1, 2, or 3 to first remember which gate it is (via a new variable "gate"), and then to set the variable "givequestion" to TRUE. You'll see later how we will use the "gate" variable to tell Authorware which gate to erase.

Now, let's add the final bit of programming that tells Authorware to check if the player's game piece is overlapping the maze exit.

☐ **Add the following 2 lines:**

```
--check to see if overlapping maze exit:
if Overlapping(IconID@"player", IconID@"maze exit") then gameover=TRUE
```

Again, the exact location of these lines of code inside of this text window do not matter.

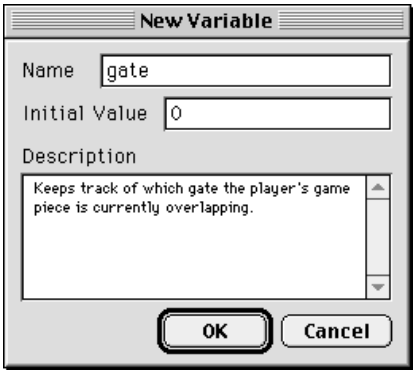
The first line is again just a comment. The second line tells Authorware to set the variable "gameover" to TRUE if the objects are overlapping. (Recall that the Decision icon "loop" is set to repeat until "gameover" is TRUE.)

That's it!

☐ **When finished, click in the close box in the Calculation icon's window; click "Yes" to save the changes to this icon.**

Authorware notices the new variable "gate" and automatically opens up a new variable dialog box for us to complete.

☐ **Type in the following initial value and description for the numeric variable "gate," then click "OK":**

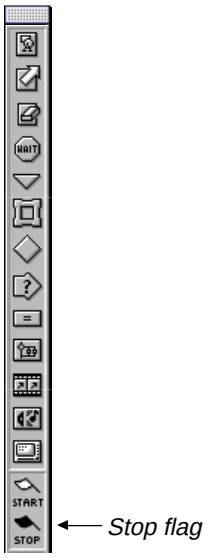


It is prudent at this point to test the program to make sure the programming logic we just constructed is working properly. If you go ahead and run the program, you will notice that nothing seems to happen when the player's game piece overlaps a maze gate. Well, this can be explained by taking a look inside the Map icon "questions" — it's empty! Even though we don't have a question yet, we can test the program with the help of the *stop flag*. You can put this stop flag anywhere you want in a program. When Authorware encounters it, an automatic pause is triggered in the program. We'll use it just to make sure that Authorware does, in fact, take the branch into the questions when the player's game piece overlaps a maze gate.

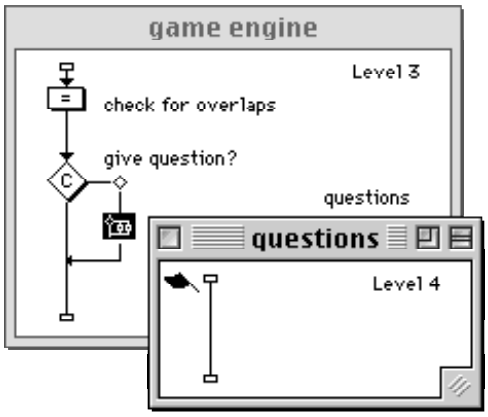
Another strategy would be to insert a temporary Display icon at the key branching point just to see if Authorware encounters it.

☐ **Double-click on the Map icon "questions"; then click "OK - Edit Map."**

☐ **Drag the stop flag to this level 4 flow chart:**



Stop flag



☐ **Restart the file.**

☐ **Maneuver the game piece to one of the maze gates.**

If the program is working properly, the Authorware will pause the file automatically (the buttons will no longer work and the author menu bar will appear at the top of the screen).

☐ **Restart the file at least two more times to test the remaining two gates.**

☐ **When finished, press Command/Control-J to go back to the flow line.**

Click once here.

☐ **Click once in the empty stop flag box to "snap" the stop flag back.**

☐ **Save the file.**

If your program is not working properly, you will need to go back and find the error.

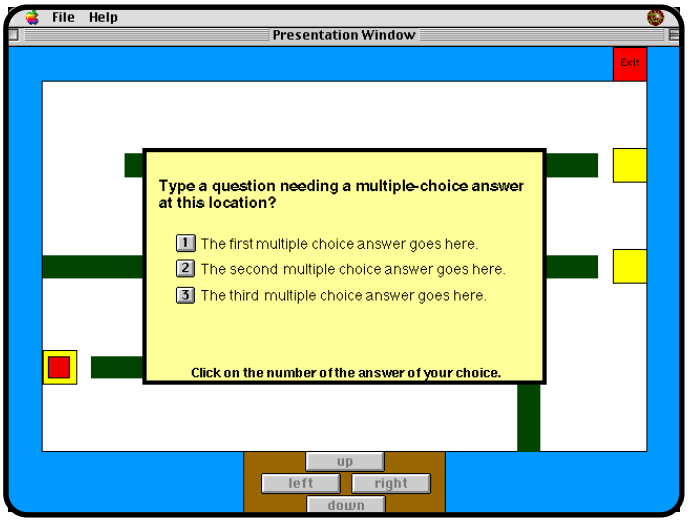
Now let's construct a question.

Constructing a generic question as a template

There are lots of ways we could set up our questions. Remember that our goal is to have the students generate questions that will be embedded into the game. Therefore, the number of questions used by the game will certainly grow over time. A good strategy is to set up the first question as a kind of template which can be copied, pasted, then modified. Ideally, we should also spend extra time getting the "look and feel" of this question just right. If we want to change the structure of the questions later, we will have to make the changes manually in *each* question. The simplest ways to set up this first question are ultimately the most labor intensive, whereas a question constructed with sophisticated programming will be easier to modify. We'll compromise and set up our question to use just a couple of simple programming tricks.

First, let's decide on the design of the question.

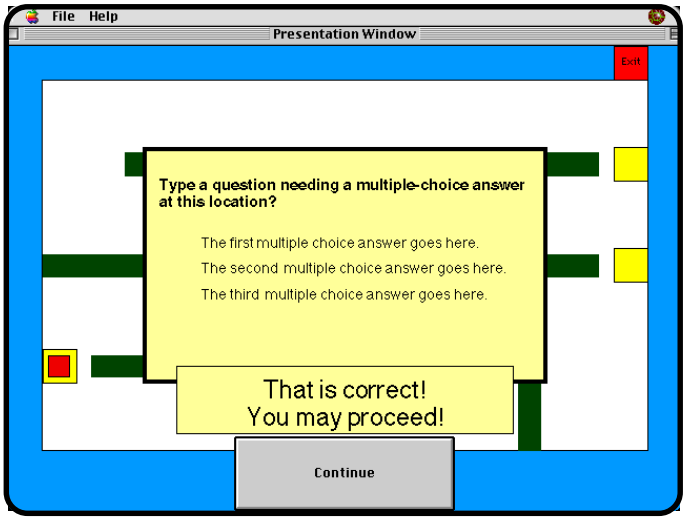
Here's a simple design that we will use:



As you can see, the question will pop up in the center of the screen. Since we don't have a "real" question to type at this point, we will just enter short phrases instructing us what we need to do later. Obviously, it will be helpful as we test the program if we know which answer is correct. So, we'll just make the first answer the correct choice in this generic question, but we will program it in such a way as to make it easy for us to change the correct answer to be any of the choices.

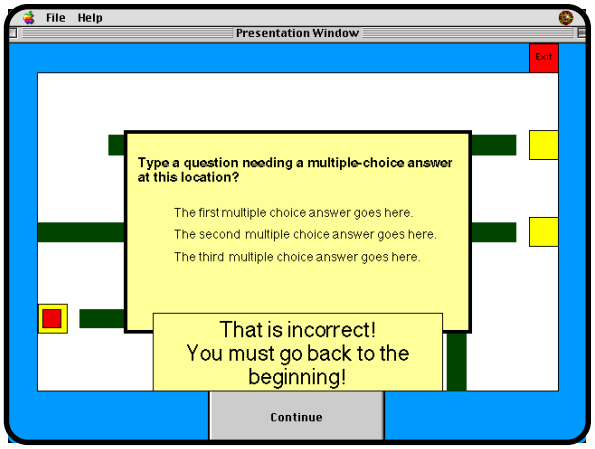
Otherwise, it won't be much of a challenge for the players!

When the player answers correctly, the following message will appear on the screen:



As you can see, the feedback is simple. Notice that we will cleverly hide the question's directions and the buttons that move the player's game object. When the player clicks "Continue," the question, feedback, *and the gate* will be erased.

On the other hand, if they answer incorrectly, the following message will appear on the screen:

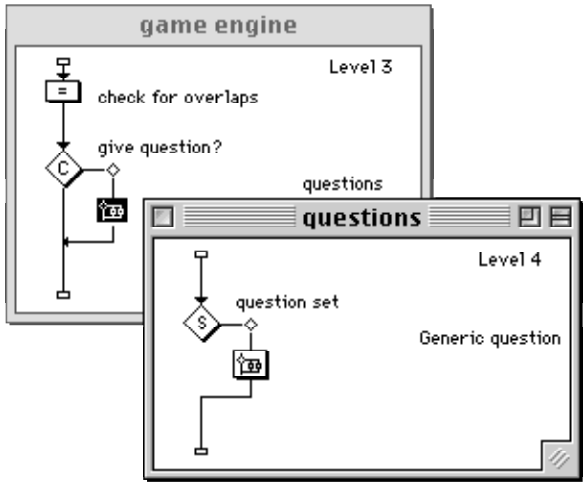


When the player clicks the continue button, the question and feedback will be erased, but the gate will remain. Also, the player's object will be magically transported back to the starting block as an additional penalty.

We begin by setting up another Decision icon to which we will eventually attach *all* of the questions.

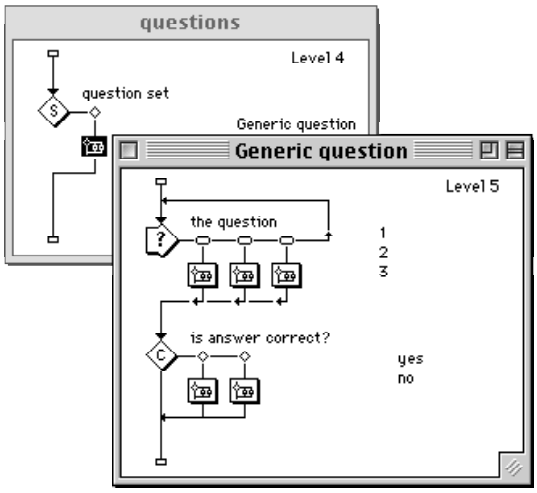
☐ **Drag a Decision icon to the Level 4 flow line inside the Map icon "questions" and title it "question set."**

☐ **Attach a Map icon to this Decision icon and title it "Generic question":**



Later, we can just copy and paste this generic question as another branch to this Decision icon. We will then have lots of flexibility in how the questions will be presented (via the branching features of the Decision icon).

Here is an overview of what we will construct:



Yes, we could just put the appropriate feedback and actions inside the branches off of the Interaction icon. However, the strategy we will use is "leaner" — we will not need to duplicate the "incorrect" feedback/actions two times (or more if we decide to have more answer choices). It will also be easier to make modifications later, if needed, to the feedback and actions.

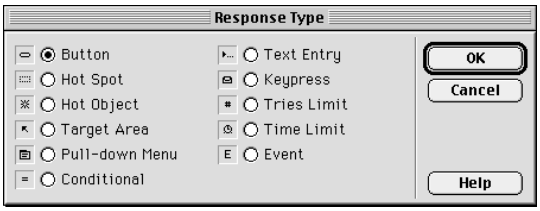
As you can see, the question will have three pushbuttons (one for each multiple-choice answer). The Decision icon "is answer correct?" with the "yes" and "no" branches will generate the appropriate feedback and actions.

☐ **Double-click on the Map icon "Generic question".**

☐ **Drag an Interaction icon to the Level 5 flow line and title it "the question."**

☐ **Attach a Map icon to this Interaction icon.**

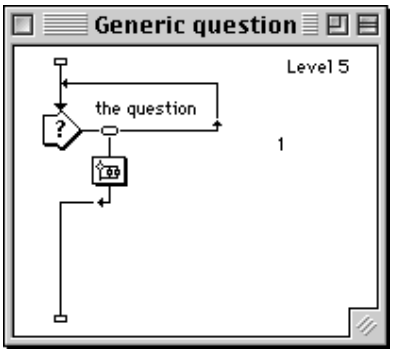
☐ **Make sure that the radio button is checked beside "Button," then click "OK":**



☐ **Title this branch "1."**

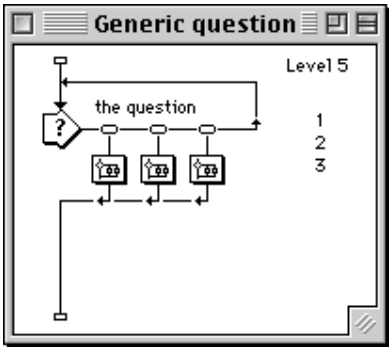
Recall that the title of a pushbutton branch *becomes* the name of the pushbutton when the file is run.

☐ **Command/Control-click on the flow line exiting the "1" branch to have the flow "exit interaction" as per the following:**



Now let's go ahead and set up the other two pushbuttons.

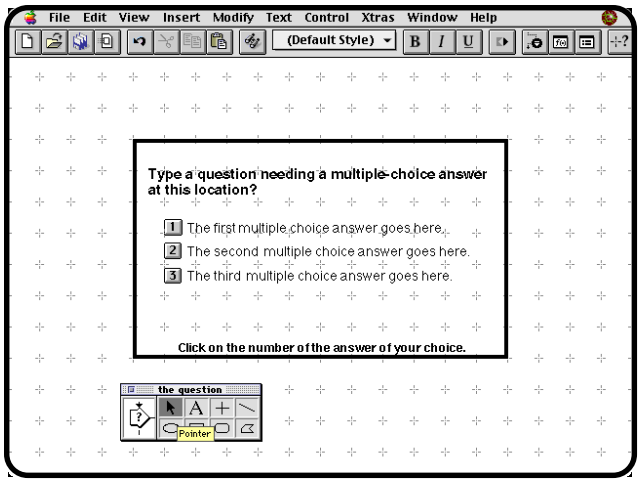
☐ **Attach two more Map icons to this Interaction icon and title them "2" and "3."**



Let's construct the question that will be displayed by this Interaction icon.

☐ **Double-click on the Interaction icon "the question".**

☐ **Using the text tool and rectangle tool, construct a display similar to the following:**



***Tip**: Don't fill in the rectangle with any pattern or color until after you are finished because they will cover the buttons.*

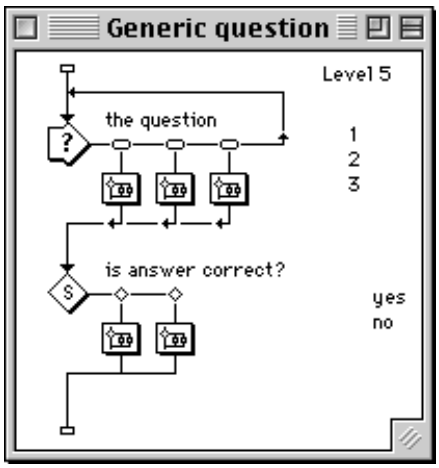
Obviously, you will need to resize the three buttons and position them carefully beside the 3 answer choices.

☐ **When finished, click in the close box in the tool box to go back to the flow line.**

We're not quite finished with this Interaction icon. But first, let's set up the feedback to be given and the actions to take place depending on whether the question is answered correctly or not.

☐ **Drag a Decision icon to the Level 5 flow line just below the Interaction icon "the question" and title it "is answer correct?"**

☐ **Attach two Map icons to this decision and title them "yes" and "no":**

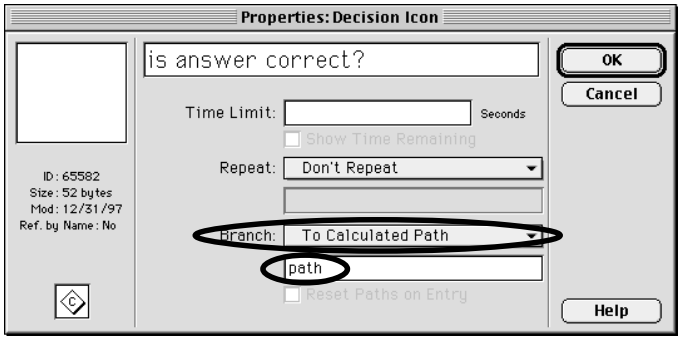


The idea here is that we want Authorware to branch to the "yes" map when the answer is correct and the "no" branch when the answer is incorrect. To do this, we will set up another variable called "path" that will work in tandem with the Interaction icon pushbuttons.

☐ **Double-click on the Decision icon "is answer correct?"**

The dialog box controlling this Decision icon's branching and repeating functions pops open.

☐ **Change the branching option from "Sequentially" to "To Calculated Path" and type "path" in the text box just below it:**

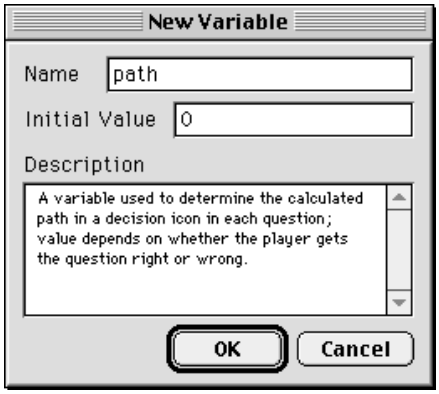


As you will see, the variable "path" will point Authorware to either the first or second branch. Recall that Authorware uses the value of whatever is inside this text box to figure out which branch to go to. We will set "path" to either a 1 or 2, depending on whether the answer to the question is correct or incorrect.

☐ **Click "OK."**

The new variable dialog box automatically pops open.

☐ **Type in the following initial value and description for the numeric variable "path," then click "OK":**



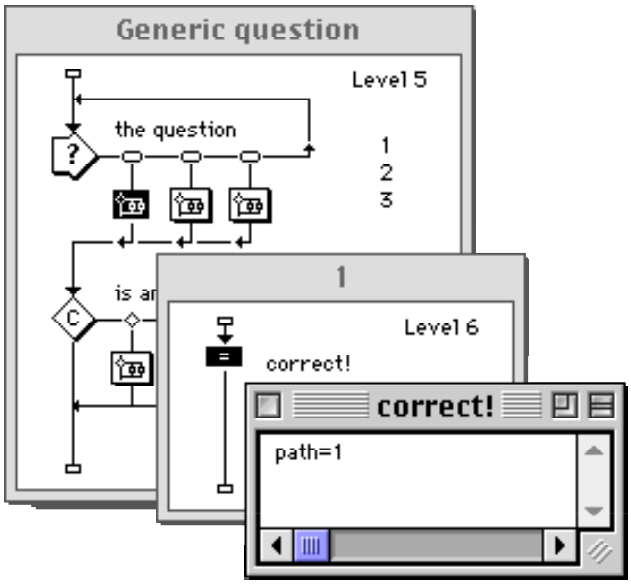
Now we set "path" to either a 1 or 2 in each of the pushbutton branches of the generic question.

☐ **Double-click on the Map icon in the first pushbutton branch titled "1."**

Recall that we decided to make answer 1 the correct answer.

☐ **Drag a Calculation icon to this Level 6 flow line and title it "correct!"**

☐ **Double-click on this Calculation icon and type in the following line in the text window:**



If the player chooses pushbutton 1 as their answer, the flow will get sucked into the first branch at which point this Calculation icon will tell Authorware to set the variable "path" to 1. Next, the flow goes out of the interaction group and down into the Decision icon "is answer correct?" at which point the variable "path" then tells Authorware to take the first branch to "yes" (because its current value is 1).

☐ **When finished, click in the close box in the Calculation icon's window; click "Yes" to save the changes to this icon.**

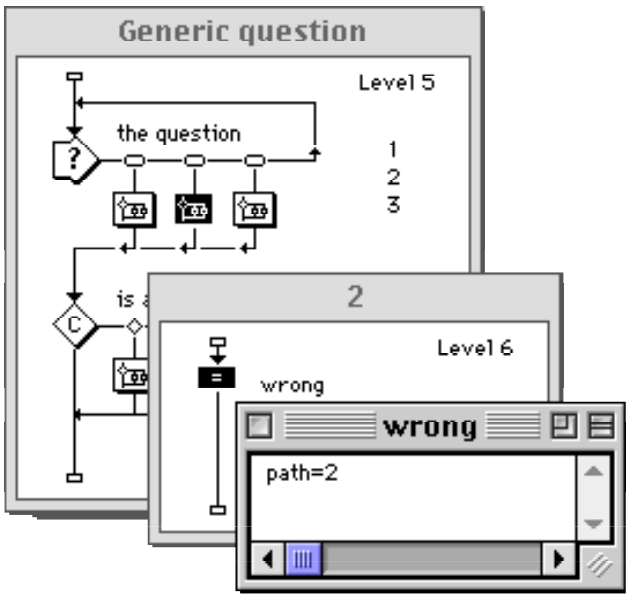
Next, we do the same thing in the other two pushbutton branches, except that we will set path to 2 instead.

☐ **Double-click on the Map icon in the first pushbutton branch titled "2."**

This will be an incorrect answer.

☐ **Drag a Calculation icon to this Level 6 flow line and title it "wrong."**

☐ **Double-click on this Calculation icon and type in the following line in the text window:**



If the player chooses pushbutton 2 as their answer, the flow will get sucked into the second branch at which point this Calculation icon will tell Authorware to set the variable "path" to 2. Next, the flow goes out of the interaction group and down into the Decision icon "is answer correct?" at which point the variable "path" then tells Authorware to take the second branch to "no" (because its current value is now 2).

☐ **When finished, click in the close box in the Calculation icon's window; click "Yes" to save the changes to this icon.**

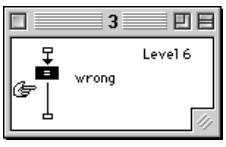
Since the third pushbutton branch is also an incorrect answer, we can just copy and paste the Calculation icon titled "wrong" into it. Since we've done this "copy and paste" many times already in previous chapters, only abbreviated instructions will be given.

☐ **Copy the Calculation icon titled "wrong."**

☐ **Close the Level 6 flow line belonging to the Map icon "2."**

☐ **Double-click on the Map icon "3" (the third pushbutton branch).**

☐ **Click once on this Level 6 flow line so that the paste hand appears.**



☐ **Paste the Calculation icon "wrong" to this Level 6 flow line.**

Adding the feedback and actions

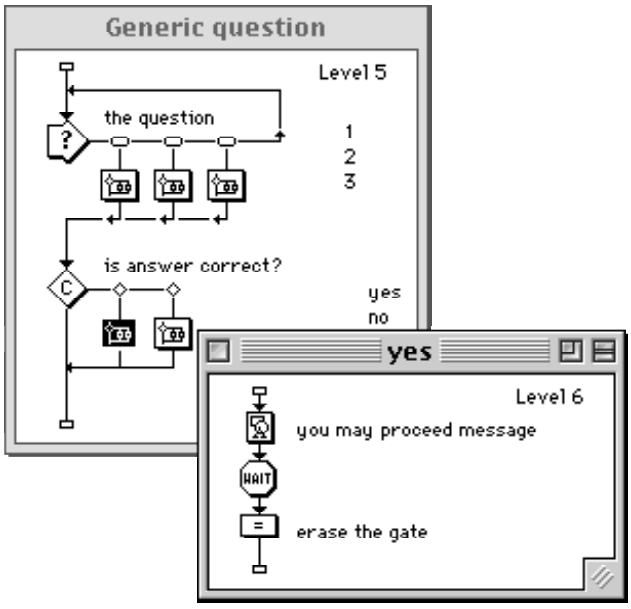
Now it is time to program Authorware to not only display appropriate feedback if the player answers the question correctly or incorrectly, but also to perform the corresponding game actions, such as erasing the gate or putting the player's game piece back to the beginning. All of this programming will be put in the "yes" or "no" Map icons attached to the Decision icon "is answer correct?"

Let's begin with the feedback and actions associated with a correct answer to the question.

☐ **Double-click on the Map icon "yes" attached to the Decision icon "correct answer?".**

We will add a Display icon and Wait icon to manage the feedback displayed on the screen followed by a Calculation icon. Recall that if the player answers the question correctly, the maze gate should erase to allow the player to move on. This Calculation icon will do this task.

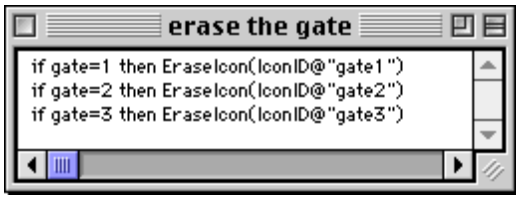
☐ **Drag a Display icon, Wait icon, and Calculation icon to this Level 6 flow line and title them as per the following:**



Let's program the Calculation icon "erase the gate" first.

☐ **Double-click on the Calculation icon "erase the gate" to open its text window.**

☐ **Type the following 3 lines:**



You can find out more about the "EraseIcon" function (or have Authorware paste it in for you) by choosing "Functions" under "Window." This function is in the category of Icon functions.

These lines use the handy "EraseIcon" function. With it, you can program Authorware to erase any Display icon with a command instead of an actual Erase icon. The tricky part is making sure that Authorware erases the right gate. Fortunately, we took care of this issue awhile back when we put in the code inside the Calculation icon "checking for overlaps." Recall that we set the variable "gate" to the number of the gate when Authorware noticed that the player's game piece was overlapping that particular maze gate display. Therefore, these three lines just tell Authorware to erase the particular maze gate if the variable "gate" is set to that number.

Getting confused? It's really pretty simple if you follow the steps we've taken up to this point.

☐ **When finished, click in the close box in the Calculation icon's window; click "Yes" to save the changes to this icon.**

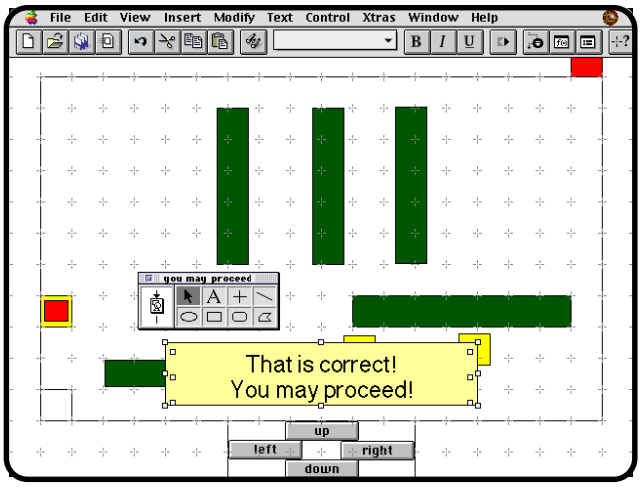
Let's program the display and Wait icons just by running the file.

☐ **Restart the file; maneuver the game piece to any of the gates to trigger the question; then answer the question correctly (choice 1).**

By answering the question correctly, you will be sent into the Decision icon branch containing the Map icon "yes" which also contains the three icons on which we are working. Since the Display icon "you may proceed message" has yet to be defined, Authorware automatically pauses the execution of the file and opens up this icon for us.

Alert! *The game objects will show behind the question, so you will need to fill in the rectangle around the question with a suitable color and set it to opaque mode. However, this will not cover the player's game piece because animated icons always show on top of nonanimated icons. A "quick fix" is to use the layering feature to make all the displays in the question appear "above." Select "Icon" under the "Modify" menu, then select "Properties", when a Display or Interaction icon is either opened or highlighted. Layering is in the Display options. Enter a positive number, such as 5. (Use different numbers to layer different icons — an icon with a layer number of 10 will be on top of an icon with a layer number of 5.)*

☐ **With the text tool and rectangle tool, construct feedback similar to the following:**



Fill in the rectangle with a suitable color and make sure it is set to opaque mode.

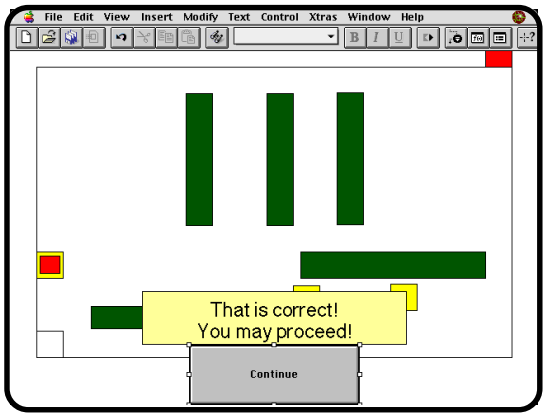
You've probably noticed that the question has disappeared. That's because the default setting for all interactions is to erase the interaction "upon exit." We'll fix that problem momentarily. For now, just construct the feedback on the screen roughly where you see it above (we can fine tune it later).

☐ **When finished, click in the close box in the tool box.**

Authorware resumes running the file. The next icon it encounters is the Wait icon. Of course, Authorware doesn't put it where we want it. No problem.

☐ **Press Command/Control-P to pause the lesson.**

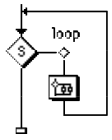
☐ **Drag the Continue button to the bottom of the screen and resize it to cover the player controls:**



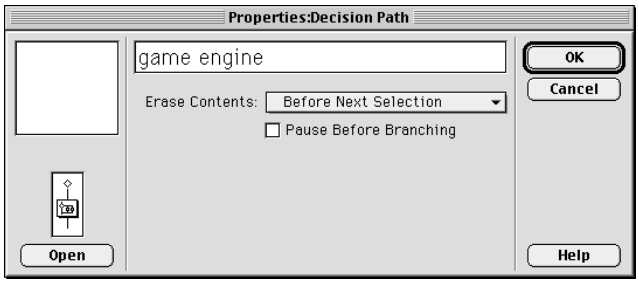
☐ **When finished, press Command/Control-P to proceed.**

Authorware resumes executing the file. It now waits indefinitely until we press the Continue button.

☐ **Click once on the Continue button.**



The feedback and the gate should erase. The feedback erases because it is contained in a branch off of a Decision icon. The default setting of a Decision icon branch is to erase "before next selection." The very first Decision icon in which all of this stuff is attached is the Decision icon "loop" way back on the Level 2 flow line. If you double-click on the small diamond just above the "game engine" branch, you will notice that in the dialog box that it is set to erase "before next selection":



This means that anything displayed inside this branch will be erased right before the next loop.

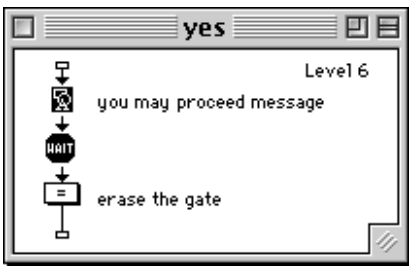
Let's work on the feedback and game action to be triggered when the player answers the question incorrectly.

☐ **Press Command/Control-J to jump back to the flow line.**

Since the feedback given when the player answers incorrectly is structured the same way as the "That is correct!" feedback, we can save some time by copying and pasting this display. We can also copy and paste the Wait icon — Authorware will remember its location and size.

Copying more than one icon can be accomplished by holding down the shift key while selecting the icons.

☐ **While holding down the shift key, click once on the Display icon "you may proceed message" and the Wait icon just below it.**



☐ **Select “Copy” from the “Edit” menu.**

This copies these two icons to the clipboard. Now we can paste these icons into the Map icon “no.”

☐ **Close the Level 6 flow line titled “yes.”**

☐ **Double-click on the Map icon “no” attached to the Decision icon “is answer correct?”.**

☐ **Click once on this Level 6 flow line so that the paste hand appears.**

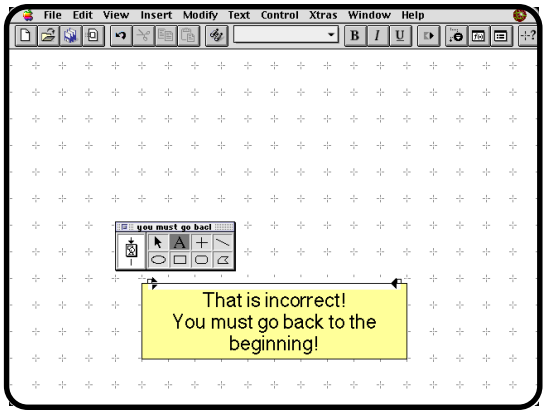
☐ **Select “Paste” from the “Edit” menu.**

☐ **Rename the Display icon “you must go back message.”**

Let’s go ahead and modify the feedback in this Display icon.

☐ **Double-click on the Display icon titled “you must go back message.”**

☐ **With the text tool, change the feedback to the following:**

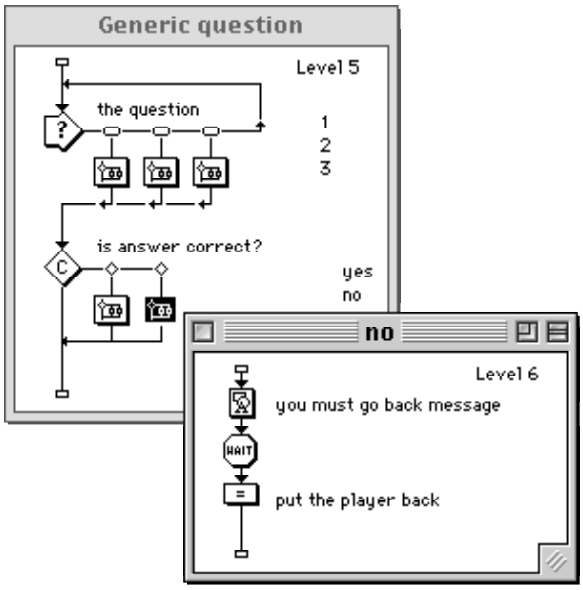


Resize the rectangle as necessary.

☐ **When finished, click in the close box in the tool box.**

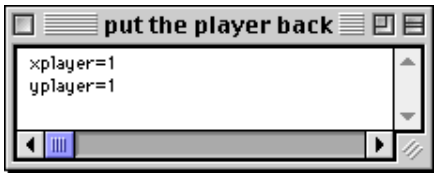
Recall that if the player answers the question incorrectly, the player’s game piece should be moved back to the starting point. A Calculation icon will do this task.

☐ **Drag a Calculation icon to this Level 6 flow line and title it "put the player back":**



☐ **Double-click on the Calculation icon "put the player back" to open its text window.**

☐ **Type the following 2 lines:**



These lines reset the two variables that control the animation of the player's game piece — "xplayer" and "yplayer" — to 1. Recall that 1,1 are the coordinates of the starting block. Since the Motion icon "animate the player" is perpetually animating the player's game piece, the moment these variables change Authorware will immediately move it to these coordinates.

☐ **When finished, click in the close box in the Calculation icon's window; click "Yes" to save the changes to this icon.**

Let's test the program to make sure these icons work.

☐ **Restart the file; maneuver the game piece to any of the gates to trigger the question; then answer the question incorrectly (choice 2 or 3).**

By answering the question incorrectly, you will be sent into the Decision icon branch containing the Map icon "no" which also contains the three icons with which we are working.

Bug alert! *If you continue to press the screen buttons or keyboard arrows, you can avoid answering the question. Don't worry, we will also fix this bug before we end the chapter.*

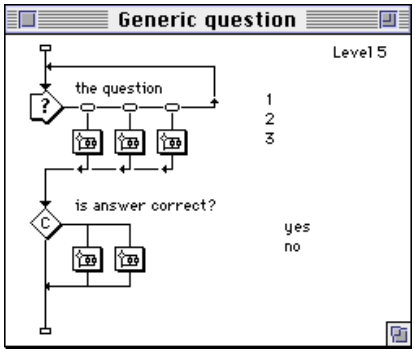
The feedback message should appear. When you press the Continue button, the feedback should erase and the player's game piece should go back to the starting point.

☐ **Press Command/Control-J to go back to the flow line.**

☐ **Save the file.**

We still need to take care of the problem associated with having the question stay on the screen with the feedback (as of now, it disappears). This is an easy problem to solve.

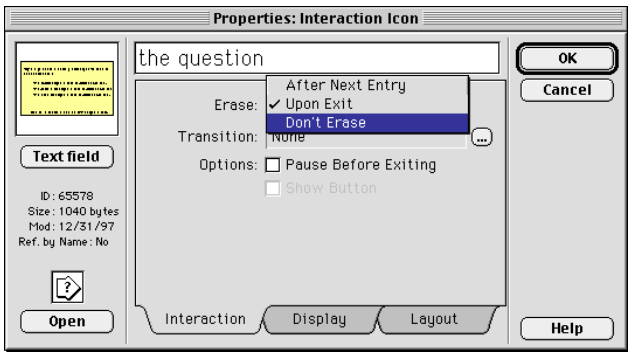
☐ **Go back to the Level 5 flow line containing the Interaction icon "the question":**



☐ **Hold down the Command/Control key, then double-click on the Interaction icon "the question."**

This opens up the dialog box for this Interaction icon. We need to make one important change. If you go to the Interaction options, you will notice that it is set to erase the interaction "upon exit." We need to change this to "don't erase" because we need to keep the question displayed on the screen while the feedback (either positive or negative) is also displayed.

☐ **In the Interaction options, change the erase option from "Upon Exit" to "Don't Erase":**



☐ **Click "OK" to go back to the flow line.**

When you run the file now, the question should stay on the screen with the feedback. Now you can fine tune the position of the feedback while the file is running.

☐ **Run the file several times to make sure all is working properly and to fine tune the position of the feedback.**

☐ **Press Command/Control-J to go back to the flow line.**

☐ **Save the file.**

Adding the timer

A unique feature of this game is that it uses the time it takes the player to successfully get through the maze as a scorekeeping feature. Although the idea of keeping track of the total time that the player spends in the maze may sound complicated, it's actually very easy to do — again, with the help of some Authorware system variables.

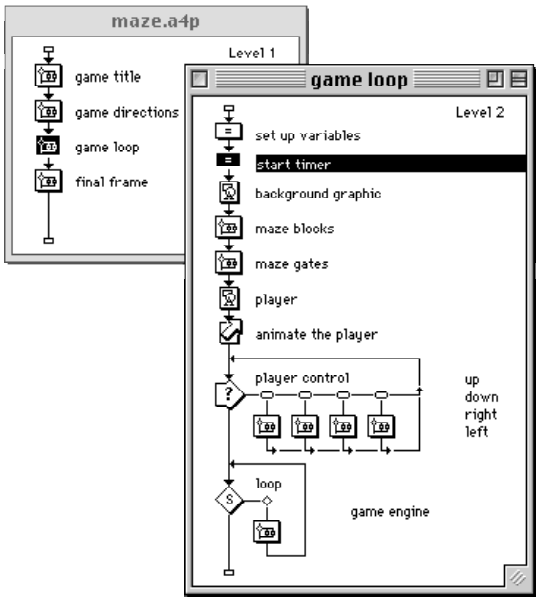
First off, you should note that Authorware has numerous built-in system variables keeping track of time spent in the file, interactions, etc. We need to start the clock running as soon as the game begins and stop it as soon as the player maneuvers their game piece to the maze exit. Therefore, we are going to construct our own variables to keep track of the starting and ending time.

An example is the system variable "TimeInInteraction" (in the category of interaction variables). This keeps track of the total time that a user spends in a specified interaction.

First, we need to decide when the game's timer should begin. It seems to make sense that the clock should start running as soon as the player enters the game loop.

☐ **Drag a Calculation icon to the Level 2 flow line inside the Map icon "game loop" to the point just below the Calculation icon titled "set up variables."**

☐ **Title this Calculation icon as “start timer”:**



☐ **Double-click on this Calculation icon to open its text window.**

☐ **Type in the following line:**

This interesting line simply converts the current time of day *to seconds* and put this value into a new variable “starttim.” “Hour,” “Minute,” and “Sec” are system variables that check the computer’s built-in clock for the current hour, minute, and second. For example, as I type this it is currently 2:35:57 on a Saturday afternoon, or, in seconds, it is 9357 on a Saturday afternoon. Think about it. There are 60 seconds in a minute, so we multiply 35 by 60 to get 2100. There are 3600 seconds in an hour, so we multiple 2 by 3600 to get 7200. Then we add these two sums to 57 for a time of 9357 seconds. What twisted reason could we possibly have for wanting to do this? Well, when the game is over, we will do the same thing, but put the value into another variable called “endtime.” If we subtract “starttim” from “endtime” we will have the total time, in seconds, that the player took to complete the maze.

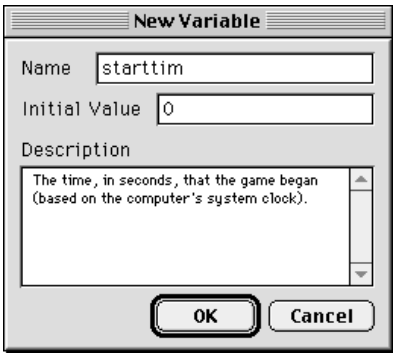
As previously noted, we can’t use the name “starttime” because it is already being used by another system variable.

To learn more about the system variables “Hour,” “Minute,” and “Sec” go to “Show variables...” under the “Data” menu. They can be found in the Time category.

☐ **When finished, click in the close box in the Calculation icon's window; click "Yes" to save the changes to this icon.**

The new variable dialog box automatically pops open.

☐ **Type in the following initial value and description for the numeric variable "starttim," then click "OK":**



Next, we need to determine when to stop the clock and compute the player's score. A good place would be first thing in the Map icon "final frame" since the flow of the program would go there as soon as the player's game piece overlaps the maze exit. (Recall that the Decision icon "loop" repeats until the variable "gameover" is true. This variable is only set to true when the player's game piece overlaps the maze exit as programmed in the Calculation icon "check for overlaps.")

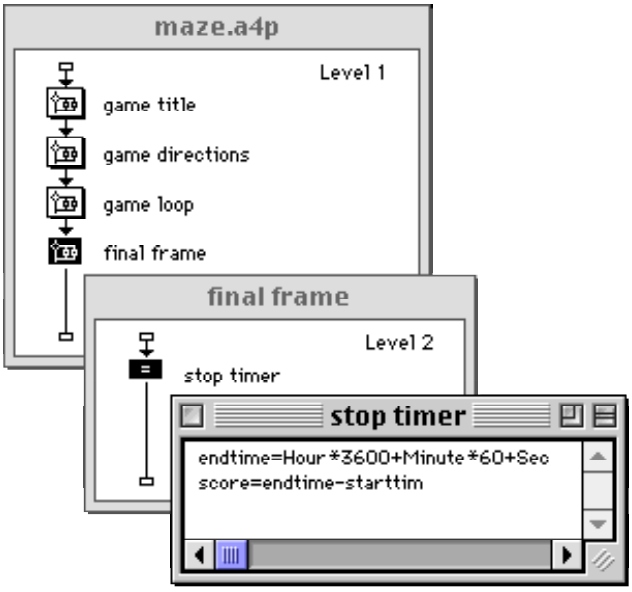
☐ **Close the Level 2 flow line titled "game loop."**

☐ **Double-click on the Map icon "final frame."**

☐ **Drag a Calculation icon to this Level 2 flow line and title it "stop timer."**

☐ **Double-click on this Calculation icon to open its text window.**

☐ **Type in the following lines:**

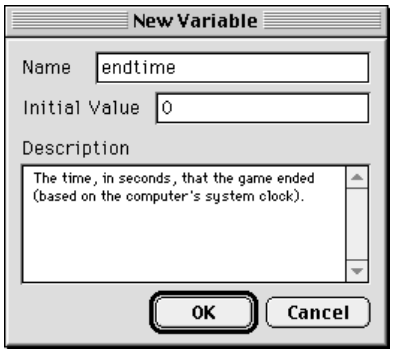


The first line is virtually identical to the one we created just a moment ago. It converts the current time of day to seconds and stores the value in the new variable "endtime." As I type this it is currently 3:02:17, or 10937 when expressed in seconds. The second line subtracts the starting time from this ending time and stores the difference in the new variable "score." So, it has taken me 1580 seconds to write this much of the book!

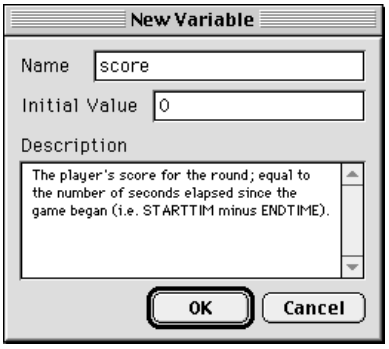
☐ **When finished, click in the close box in the Calculation icon's window; click "Yes" to save the changes to this icon.**

The new variable dialog box automatically pops open for each of these two new variables.

☐ **Type in the following initial value and description for the numeric variable "endtime," then click "OK":**



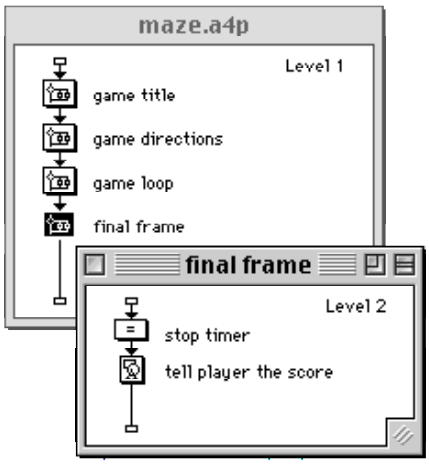
☐ **Type in the following initial value and description for the numeric variable "score," then click "OK":**



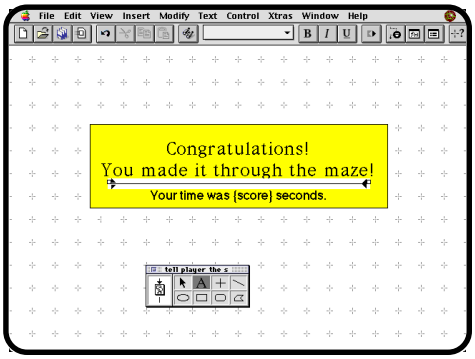
Let's run the file to make sure these timing variables are working.

Let's add a Display icon right after this Calculation icon to display the player's score.

☐ **Drag a Display icon to this Level 2 flow line and title it "tell player the score."**



☐ **Double-click on the this Display icon and construct the following message with the text and rectangle tools:**



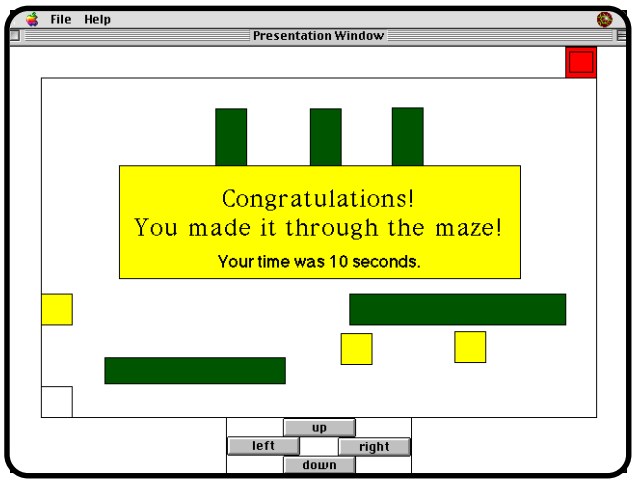
As explained in previous chapters, one displays the contents of a variable by typing the variable name inside a set of braces, such as {score}.

When you click on the pointer (in the tool box), the current value of the variable will be displayed (probably 0 in this case).

Let's test it.

☐ **Run the file and maneuver the player's game piece to the maze exit.**

At that point, you should get the following message displayed on the screen containing the total number of seconds it took you to get to the maze exit:



☐ **Press Command/Control-J to jump back to the flow line.**

☐ **Save the file.**

Debugging the game

The game is working pretty well at this point. Still, there are three small annoying problems. The first is that if you press the arrow keys fast enough the player's game piece goes right "over" the maze blocks or maze gates. Second, you can avoid answering questions by continuing to press the button controls or arrow keys. Third, if you press the buttons or arrow keys repeatedly and then stop, the computer seems to "remember" how many times the button or key was pressed and continues to move the game piece until it "catches up." Let's fix these three problems, starting by "disarming" the game buttons.

Go back and review the two "Bug Alerts!" — one in the section "Setting up the game engine loop" and the other in the section "Constructing a generic question as a template."

"Disarming" the game buttons

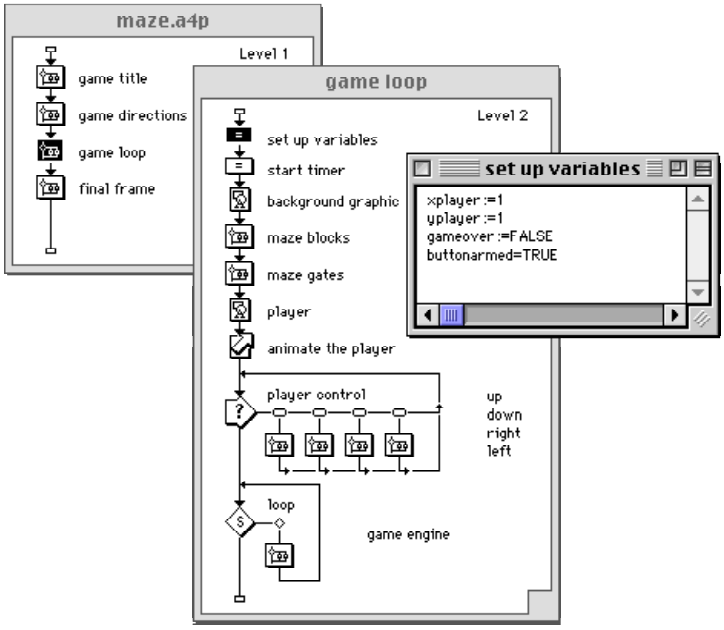
One solution to the problems of the game piece "running over" the maze objects without triggering the appropriate action and avoiding having to answer the questions is to deactivate or "disarm" the buttons for a moment to give Authorware adequate time to execute all of the programming in the Calculation icon titled "check for overlaps." We will accomplish this by setting up one more logical variable for the game called "buttonarmed." When it is TRUE, the buttons (or keyboard equivalents) are active and ready to be pressed. When FALSE, they prevent the player from moving the game piece. We will set this variable to FALSE as soon as any button is pressed and then reset it to TRUE only at the very end of the game loop. This will give Authorware a chance to complete the entire cycle for the loop.

This is the last variable we will need in this game.

Let's go ahead and set it up (to TRUE) in the Calculation icon titled "set up variables" at the start of the game loop.

☐ **Double-click on the Map icon "game loop" located on the Level 1 flow line.**

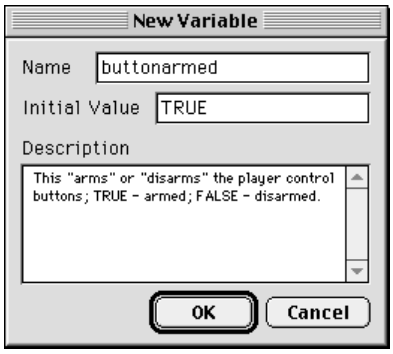
☐ **Double-click on the Calculation icon "set up variables" and add the statement "buttonarmed=TRUE" in the text window:**



☐ **When finished, click in the close box in the Calculation icon's window; click "Yes" to save the changes to this icon.**

The new variable dialog box automatically opens.

☐ **Type in the following initial value and description for this variable; then click "OK":**



Next we need to set this variable to FALSE inside the Map icon of *each* button branch and then tell Authorware to allow each button to be "active" only when this variable is TRUE. This way, as soon as the player presses a button, all the buttons will become inactive until the end of the loop containing the game engine (at which point we will set this variable to TRUE).

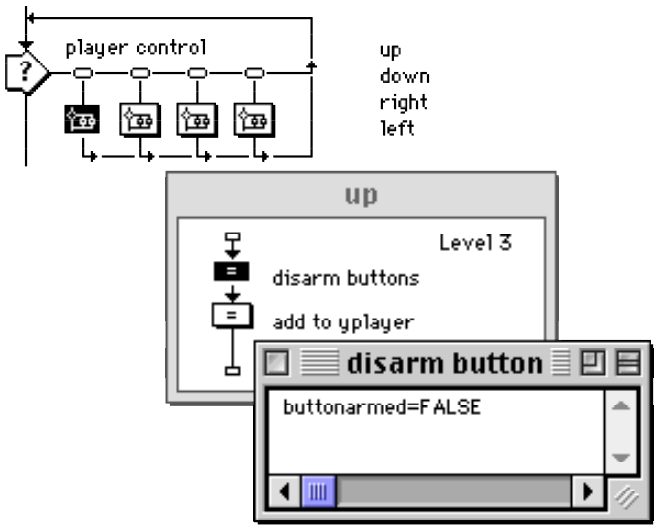
Let's start with the UP button.

☐ **Double-click on the Map icon titled "up."**

☐ **Drag a Calculation icon to the <u>top</u> of this Level 3 flow line and title it "disarm buttons."**

Alert! It is very important that this Calculation icon be first inside this flow line. The reason is that we want the button deactivated before the game piece is moved. Otherwise, we will not correct the problem.

☐ **Type the statement "buttonarmed=FALSE" into the text window:**



☐ **When finished, click in the close box to close the calculation's window, then click "yes" to save the calculation changes.**

We need to put this same Calculation icon inside the three remaining button branches. We can use copy and paste to do this.

☐ **Copy the Calculation icon “disarm buttons.”**

☐ **Paste this Calculation icon inside the other three button branches (belonging to the Map icons “down,” “right,” and “left”).**

Remember to put this Calculation icon at the top of each Map icon’s flow line.

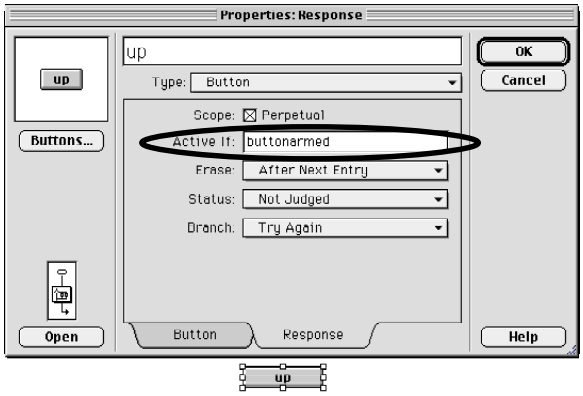
Next we need to tell Authorware to only make each of these buttons active only when the variable “buttonarmed” is TRUE. Again, we’ll start with the button titled “up.”

☐ **Double-click on the baby pushbutton just above the branch leading into the Map icon “up.”**

This pops open the dialog box containing the button options for this branch.

☐ **View the Response options by clicking on the Response tab.**

☐ **Type the variable name “buttonarmed” into the text box beside“Active If”:**



This means just what it says. This pushbutton (and its keyboard equivalent) will not function unless “buttonarmed” is TRUE.

☐ **Click “OK.”**

Let’s make this modification to the other three pushbuttons.

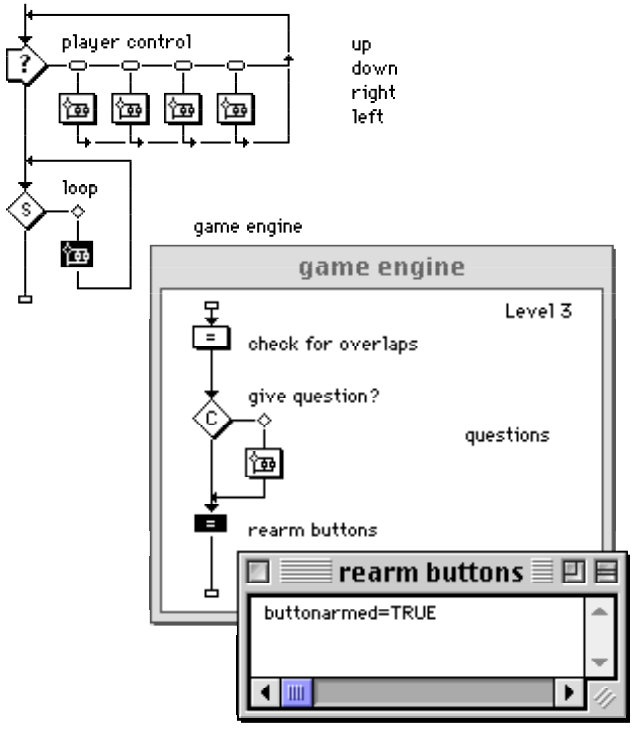
☐ **Open the Button Options dialog box for the remaining three pushbuttons (“down,” “right,” and “left”) and enter the variable name “buttonarmed” in the text box beside the phrase “Active If:”; Click “OK” when finished with each one.**

The last thing we need to do is "rearm" (or reactivate) the buttons at the very end of the "game engine." That is, we only want to let the player press one of the buttons again *after* Authorware has finished executing all of the programming for the game loop.

☐ **Double-click on the Map icon "game engine" (attached to the "loop" Decision icon).**

☐ **Drag a Calculation icon to the bottom of this Level 3 flow line and title it "rearm buttons."**

☐ **Double-click on this Calculation icon and type the statement "buttonarmed=TRUE" in the text window:**



☐ **When finished, click in the close box of the Calculation icon's window; click the "Yes" button to save the changes to this icon.**

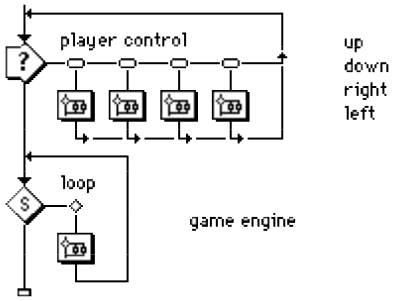
OK, we are finally ready to test the file to make sure the button "disarming" and "rearming" works.

☐ **Restart the file to test whether you can "run over" any of the game objects without triggering the appropriate action or avoid answering questions.**

Telling Authorware to ignore unprocessed keystrokes

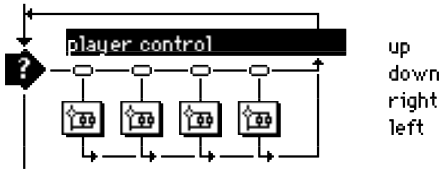
Now let's fix the last bug in the program. As is, if the player chooses to use the keyboard equivalent of any of the buttons and presses the key in rapid succession, Authorware will hold in its memory each of these keystrokes. When the player stops pressing the keys, Authorware continues to execute all of the keystrokes until it "catches up." Although there might be some contexts where this is appropriate, it will probably serve to confuse the player in this game. Fortunately, there is another function — FlushKeys ()" — that instructs Authorware to ignore any keys that have been typed by the player but not yet processed.

The key question, once again, is where to embed this function? Notice that each of the buttons loop around and go "through" the Interaction icon titled "player control" before going down to the loop containing the game engine:

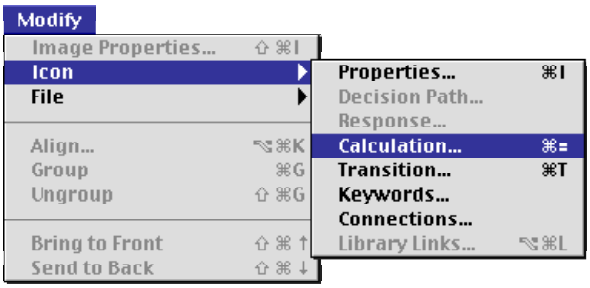


We can take advantage of this fact by embedding a calculation directly *into* the Interaction icon "player control." In fact, calculations (i.e. programming) can be embedded into any icon with the following procedures.

☐ **Click once on the Interaction icon "player control":**



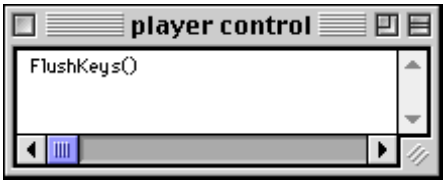
☐ **Select "Icon" under the "Modify" menu, then select "Calculation...":**



This opens a calculation text window just like the ones we have been working with.

☐ **Type "FlushKeys ()" inside this text window:**

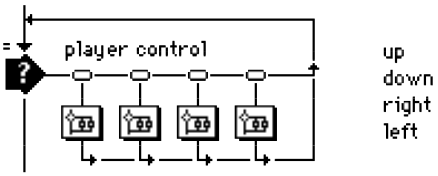
You could also have Authorware paste it in for you. Go to "Functions" under the "Window" menu; it is in the general category of functions.



☐ **When finished, click in the close box in the calculation's text window; click "Yes" to save the changes to this calculation.**

Notice the miniature equal sign appearing next to the Interaction icon signifying that a calculation is embedded in this icon:

To delete an embedded calculation such as this, you have to first open the calculation's text window (by highlighting the icon, selecting "Icon" from the "Modify" menu, then selecting "Calculations..."). Delete all of the text inside the window, then close the empty text window.



This calculation will be executed by Authorware every time this icon is encountered, which happens to be immediately after either a button or its keystroke equivalent is pressed.

☐ **Restart the file to test it.**

As you test your file, you will notice a obvious pause after each button or key is pressed.

☐ **When done testing, press Command/Control-J to jump back to the flow line.**

☐ **Save the file.**

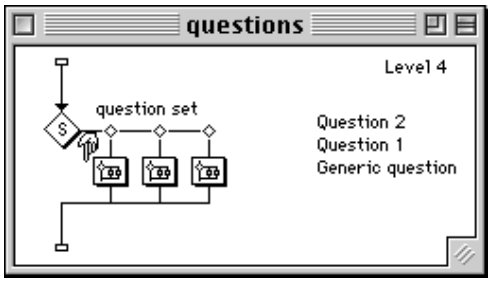
☐ **Package the file.**

See Appendix A for background and instructions on packaging.

Next Steps

Assuming the game is functioning appropriately at this point, here are the next steps to finishing the game:

1. Copy, paste, and modify the generic question for each student generated question (from their studies of World War II) you choose to embed into the game. Remember to put the Calculation icon "correct!" in whatever answer branch is the correct choice and the Calculation icon "wrong" in *all* of the incorrect answer branches. I suggest pasting each new question in front of the previous question. This will keep the generic question at the end, as per the following example:



   In this case, the questions will be sequentially presented starting with "Question 2" and ending with "Generic question." You might also explore the possibility of having Authorware present these questions in random order (double-click on the Decision icon "question set" to explore the options). Eventually of course, you would delete the Map icon "Generic question.'

2. Modify the maze layout. Recall that you can put the five maze blocks, three maze gates, and maze exit *anywhere* you want on the screen in *any* rectangular shape you want.

3. You could also change the rectangular graphics of the player's game piece and maze gates to something more aesthetically appealing. Just remember that Authorware's overlapping function continues to "see" these graphics in the rectangular area in which they are encased.

4. Finally, don't forget to add the "final frame" similar to the previous two chapters in which you give the player the option to either play the game again or quit.

Summary

In this chapter you created a fully functioning computer game in which the player maneuvers a game piece through a maze. The maze was set up in such a way as to make modifying the maze layout quick and easy leading to virtually an unlimited number of possible maze layouts. Three maze gates were also embedded in the maze layout at which point the player must answer a question successfully to move on. You constructed a generic question that can be easily copied, pasted, then modified. In this way an unlimited number of questions can be built and presented. You learned how to program Authorware to provide appropriate feedback and action depending on the player's answer. If correct, the maze gate is erased and the player continues. If incorrect, the player must go back to the starting position. All of the maze objects work via Authorware's overlapping function. You also learned how to have the computer keep track of the total time the player took to get through the maze (in seconds). This time is used as the player's score for the round and is displayed at the end of the game.

Other projects

1. Add a context to the maze that matches the nature of the questions. For example, the player's game piece could be changed to a graphic of an Allied infantry soldier trying to make his way across a battlefield strewn with barbed wire (i.e. maze blocks) to deliver an important message to the field headquarters (maze exit). The maze gates could become checkpoints.

2. Add more maze objects and more maze gates to the game to make more intricate maze layouts.

3. Add more maze exits to the layout, perhaps with each leading to different consequences (such as another maze).

4. Randomize the placement of the maze objects each time the players starts the game.

Chapter 6: How'd They Do That?

Special techniques to enhance your Authorware programs

This chapter presents a series of special programming techniques. The chapter is divided into six sections demonstrating how to add the following enhancements to your programs:

1. Allowing the user to turn the program's sound on and off.
2. Rotating screen objects by manipulating PICS movies.
3. Building a "click and hold" interaction response type (to control a QuickTime movie).
4. Building "floating response palettes" that users can move freely around the screen.
5. Creating "slide bars" that control data functions.
6. How to save a program's data in a format that can be opened and analyzed by a spreadsheet.

This chapter is designed for experienced Authorware programmers, such as those who have successfully completed the preceding chapters. There is no special sequence to learning these techniques. It's expected that you will pick and choose from the techniques depending on your needs. Unlike the other chapters in which you built complete projects, this chapter presents a series of small demonstration projects designed to quickly illustrate the concept. It is expected that once you see how to create the enhancement (by completing these small demonstration projects), you will be able to add them to your other projects. However, just like the other chapters, the sections in this chapter guide you step-by-step in building a fully functional program. This chapter does take some shortcuts, however. For example, it is expected that you know by now how to build a display and are familiar with the many different dialog boxes associated with each of the icons. This chapter will not prompt you through each mouse click for these common procedures.

Technique 1. Giving the user control over sound

Introduction

Consider the following situations:

> A fifth grader is working independently on a computer math game in the back of the room while the rest of the class is quietly reading. Every time the student answers a math question correctly, a loud rocket blast sound fills the room startling the other students (some think its funny and others are getting annoyed).
>
> A corporate executive is attending an all-day board meeting. To fill some of the long stretches of budget reports that have nothing to do with her department, she takes out her laptop and begins a computer-based tutorial of a new financial analysis application she just bought. However, heads turn her way each time the computer program plays a loud school bell sound signaling the start of new lesson section.

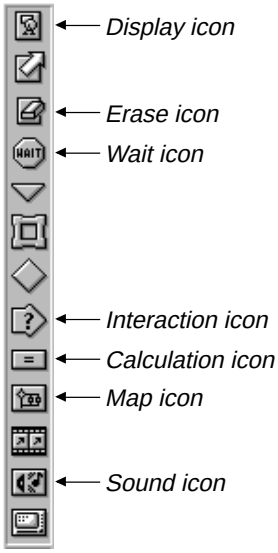
The point of these stories is simply that there are plenty of times when people use a computer software package in a situation that the designer did not intend or anticipate. Sound is a very popular feature of multimedia software, but sometimes it is necessary to turn off inconsequential sounds. While users can certainly choose to turn off all sounds via their computer's control panels, a better strategy is to embed this control within the program itself, thereby giving the user greater flexibility. This section demonstrates an easy method for doing just that in your Authorware files. This technique is also easily added to your programs after development has begun.

Getting started

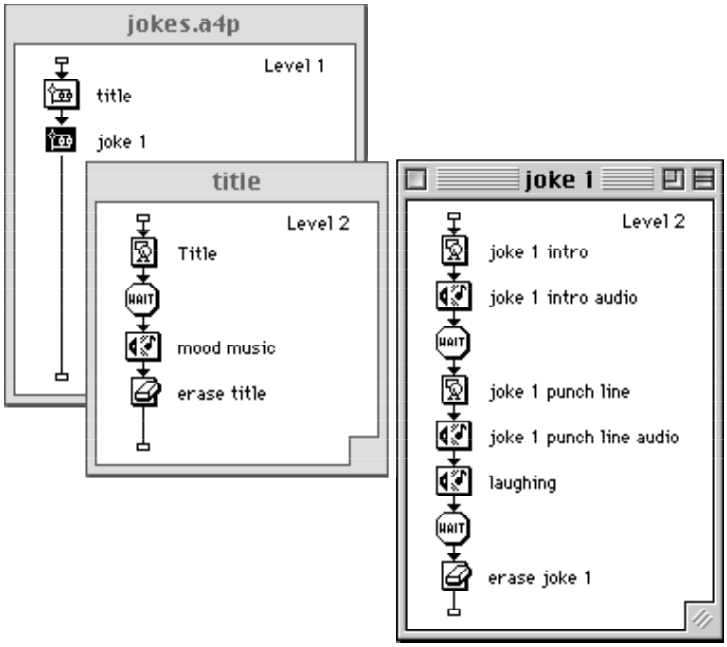
☐ Open any Authorware file in which you want to give the user control over audio played via Sound icons.

For example, consider the following Authorware program that tells a science joke via text and audio. There is also some "mood music" at the beginning and "laughter" at the end for a total of four Sound icons.

Here are the icons you will be using in this section:

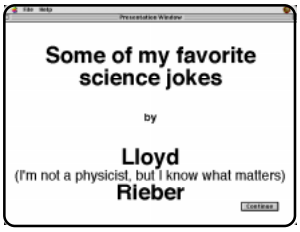


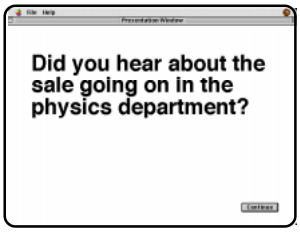
Here's the Authorware flow chart:

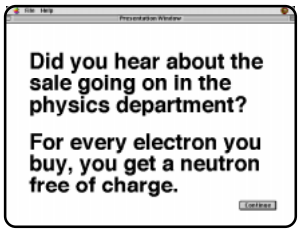


A copy of this "Science Jokes" file is available in the Rieber Clip Media in case you don't have a similar Authorware file to modify. Of course, you should also be able to quickly construct this little demonstration.

As you can see, there are four Sound icons, each playing at the appropriate time after each of the program's three displays:







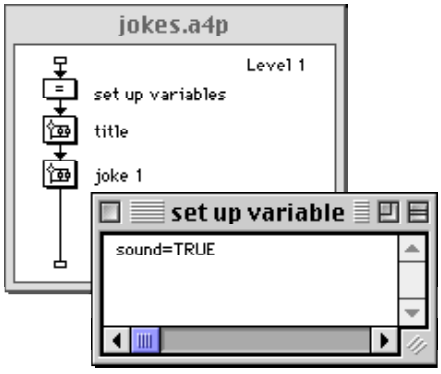
This section will demonstrate how to modify this program to give the user the option of turning off all of the audio. The steps that follow can be used to modify any Authorware program that uses Sound icons in this way.

The technique involves embedding a logical variable, "sound," in each Sound icon. When the variable is TRUE, the Sound icon will play the audio file, but won't when the variable is FALSE.

Let's start by setting up the variable "sound."

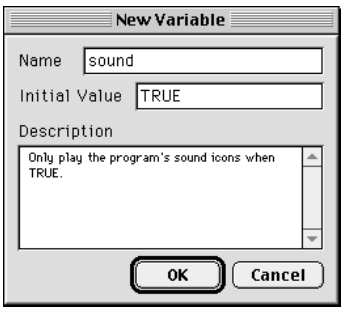
☐ **Drag a Calculation icon to the top of the flow line and title it "set up variables."**

☐ **Open the Calculation icon and insert the line "sound=TRUE":**



☐ **When finished, click in the close box in the Calculation icon's window; click "yes" to save the changes to this icon.**

☐ **Type in the following initial value and description for the logical variable "sound," then click "OK."**

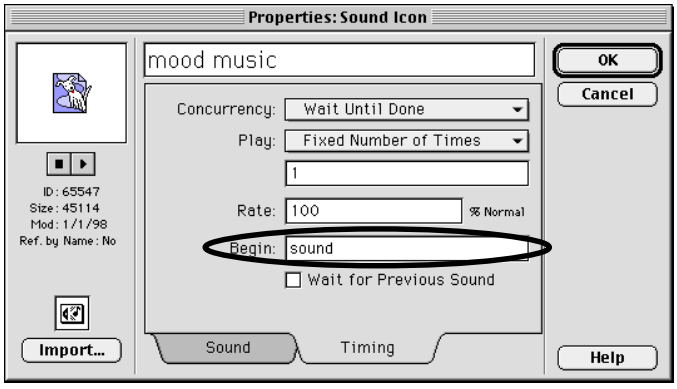


Next, we need to embed the variable "sound" into each Sound icon. Let's start with the first Sound icon "mood music."

☐ **Double-click on the Sound icon "mood music" to open the options dialog box for this icon.**

☐ **View the Timing options by clicking once on the Timing tab.**

☐ **Type the variable "sound" in the text window beside "Begin:"**



Remember that the variable "sound" is a logical variable that will contain the value of either TRUE or FALSE. Therefore, the mood music will only begin playing when it is set to TRUE.

☐ **Click "OK."**

We just need to insert the variable "sound" into each of the Sound icons we want the user to control.

☐ **Open each of the other Sound icons in the file and type the variable "sound" in the text window beside "Begin:"; then click "OK."**

Be sure you are viewing the Timing options of each Sound icon's dialog box.

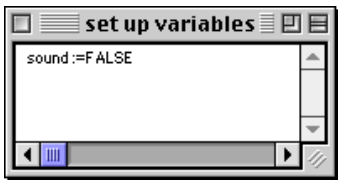
Let's test it. The first time you run the file, the audio files should play as usual since the variable "sound" is currently set to TRUE.

☐ **Restart your file to make certain that the Sound icons still play the audio files.**

Now let's test whether changing the value of the variable "sound" to FALSE prevents the audio from playing.

☐ **Jump back to the flow line.**

☐ **Double-click on the Calculation icon "set up variables" and change TRUE to FALSE in this line of code:**



☐ **When finished, click in the close box in the Calculation icon's window; click "yes" to save the changes to this icon.**

By changing this variable to FALSE, none of the audio files will play. Let's test it.

☐ **Restart the file to make certain that the audio files do not play.**

Before we save this file, let's set the variable "sound" back to TRUE (it makes more sense to have the default choice set to the sound playing).

☐ **Double-click on the Calculation icon "set up variables" and change FALSE back to TRUE.**

☐ **When finished, click in the close box in the Calculation icon's window; click "yes" to save the changes to this icon.**

☐ **Save your file.**

Giving the user control over the sound

Setting up a logical variable and embedding it into each Sound icon was easy. However, we still need to program control of this function over to the user. We'll choose to do this with a perpetual pull-down menu.

☐ **Drag an Interaction icon to the top of the program at a point just below the Calculation icon "set up variables"; title it "Sound."**

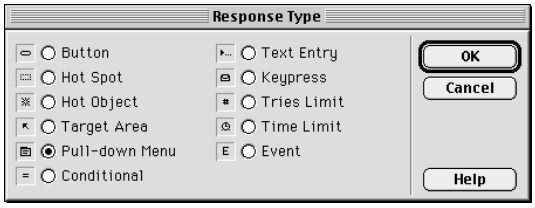
The name of the Interaction icon becomes the heading for the menubar category and the branch names become the list of options.

Next, we need to attach two Map icons to this Interaction icons with titles of "Turn sound off" and "Turn sound on." Let's set up the first one with all of the features we need, then we can copy, paste, and modify it.

☐ **Attach a Map icon to this Interaction icon.**

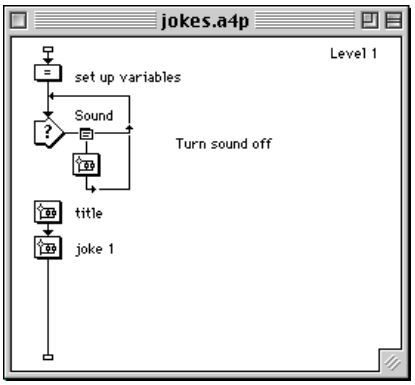
The Response type dialog automatically pops open.

☐ **Choose “Pull-down menu”; click “OK”:**



☐ **Title this Map icon (or branch) “Turn sound off.”**

When done, your flow line should look like this:



Let’s change this to a *perpetual* interaction.

☐ **Double-click on the response type symbol (the “baby pull-down menu”) just above the Map icon “Turn sound off.”**

This opens the Options dialog box for this response branch.

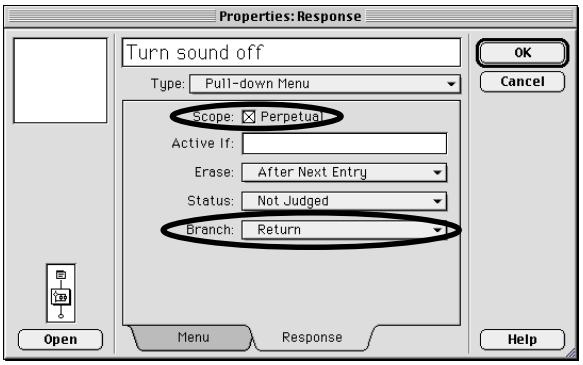
☐ **View the Response options by clicking once on the Response tab.**

☐ **Click in the check box beside “Perpetual.”**

While we are here, let’s also change the branching to “Return.”

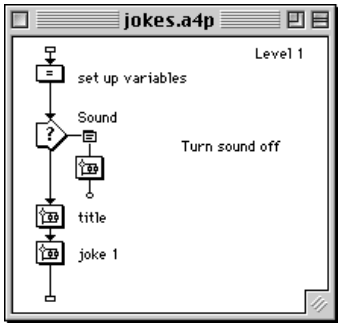
☐ **Change the Branch option from “Try Again” to “Return.”**

When done, the Options dialog box should look like this:



☐ **Click "OK."**

You should now be at the flow line:

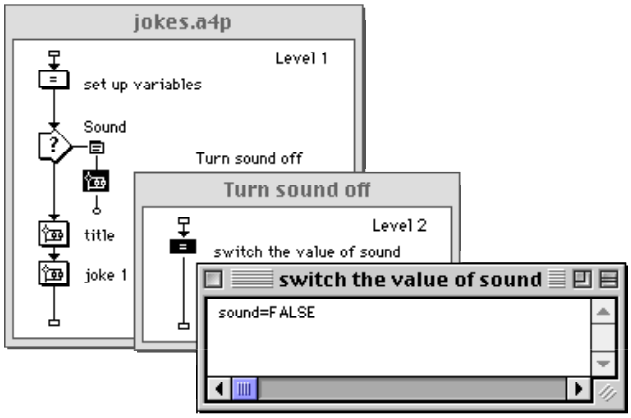


Next, we need to open this Map icon and insert a Calculation icon which will set the variable "sound" to FALSE.

☐ **Double-click on the Map icon "Turn sound off."**

☐ **Drag a Calculation icon to this Level 2 flow line and title it "switch the value of sound."**

☐ **Double-click on this Calculation icon and type "sound=FALSE" in the text window:**

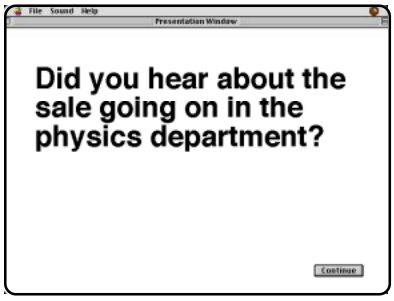


☐ **When done, click in the close box of the Calculation icon's window; click "Yes" to save the changes to this icon.**

Let's take a moment and consider what this does. Recall that the variable "sound" is initially set to TRUE. If the user selects this pull-down menu choice, the value of sound resets to FALSE, then Authorware returns to the original point in the program and resumes execution. None of the Sound icons will play their audio files when the variable "sound" is set to FALSE.

Let's test it.

☐ **Restart the file; step through the program but stop just before the punch line is given:**



Notice the new pull-down menu choice titled "Sound."

Up to this point, Authorware should have played each of the preceding audio files just like before. But now let's turn the sound off before getting the punch line.

☐ **Select "Turn sound off" from the "Sound" menu.**

This sets the variable "sound" to FALSE. If you remembered to set the Map icon's branch to "Return," the screen should remain the same.

☐ **Click "Continue."**

The Sound icon containing the punch line does not play its audio file.

Turning the sound back on

We will repeat the procedure of designing another pull-down menu choice, but this time we will set the variable "sound" to TRUE.

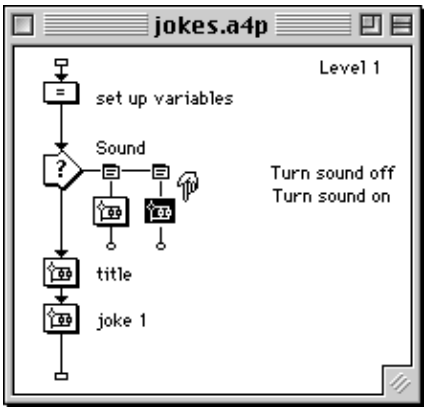
☐ **Jump back to the flow line.**

☐ **Copy the Map icon "Turn sound off."**

☐ **Click once just to the right of this Map icon so that the paste hand appears.**

☐ **Select "Paste" from the "Edit" menu** (or press "Command/Control-V)**.**

☐ **Change the name of this second pull-down menu branch to "Turn sound on":**

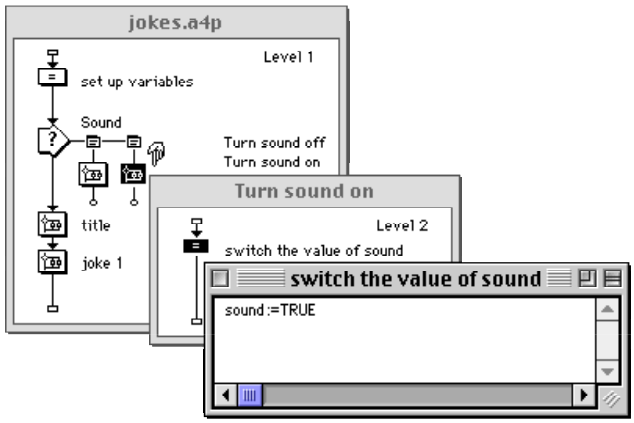


We need to change the one line of code in the Calculation icon inside this Map icon.

☐ **Double-click on the Map icon "Turn sound on."**

☐ **Double-click on the Calculation icon "switch the value of sound."**

☐ **Change FALSE to TRUE:**



☐ **When finished, click in the close box in the Calculation icon's window; click "Yes" to save the changes to this icon.**

Once again, let's test the file.

☐ **Restart the file several times to test it by alternatively selecting to turn the sound off and on via the pull-down menu.**

You should now be able to control the program's sound at will.

☐ **Jump back to the flow line.**

☐ **Save the file.**

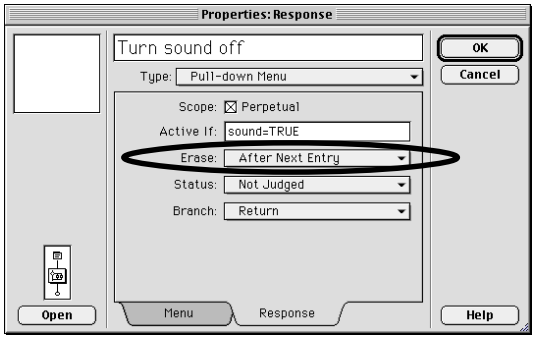
Improving the user interface

We could stop at this point. The program works well. However, there is no way for the user to tell whether or not the sound is currently on or off. Fortunately, there is a very easy way to improve this aspect of the user interface.

☐ **At the flow line, double-click on the response type symbol (the "baby pull-down menu") just above the Map icon "Turn sound off."**

The Response Type Options dialog box appears.

☐ **Be sure you are viewing the Response options, then type "sound=TRUE" in the text box beside "Active If:".**



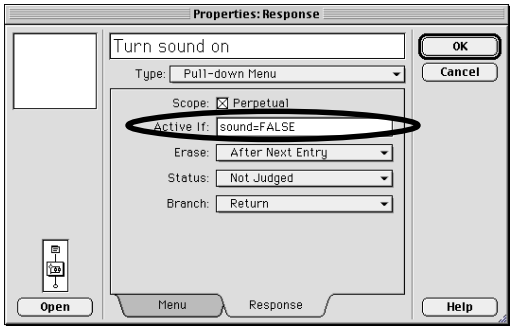
This means just what it says. This pull-down menu will not be active unless the current value of the variable "sound" is TRUE, otherwise, this pull-down menu choice will be inactive (and appear dimmed in the pull-down menu).

☐ **Click "OK."**

This sends you back to the flow line. Let's repeat the procedure for the other pull-down menu choice, except this time we will make it active only when "sound=FALSE."

☐ **At the flow line, double-click on the response type symbol (the "baby pull-down menu") just above the Map icon "Turn sound on."**

☐ **Be sure you are viewing the Response options, then type "sound=FALSE" in the text box beside "Active If:"**



☐ **Click "OK."**

☐ **Restart the file.**

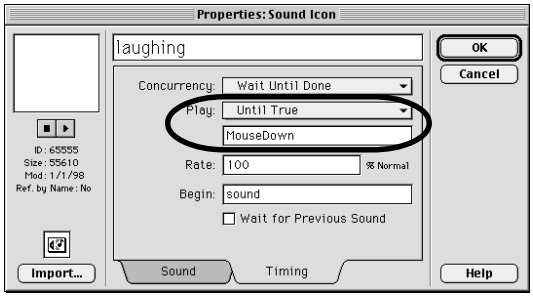
When you click and hold on the "Sound" menubar choice, you will now notice that only one is active. The other is dimmed because it is not an option.

☐ **Test the program thoroughly to be sure it works properly, jump back to the flow line.**

☐ **Save the file.**

One last sound trick

The final Sound icon in my Science Joke program includes a special trick. As you know, this icon acts as my "laugh track" immediately following the joke's punch line. To make the laughter continue for as long as I want, I modified this Sound icon's options so that the sound (i.e. laughter) will play repeatedly until I click the mouse button. To do this, go to the Timing options, then select "Until True" beside "Play:" and enter the system variable "MouseDown" in the text window:



The "MouseDown" is TRUE only when the user is pressing the mouse button (or the left button on Window's computers). In other words, the laughter continues indefinitely until the mouse button is pressed.

This is one way I use technology to build my self-esteem!

Technique 2. Rotating screen displays

Introduction

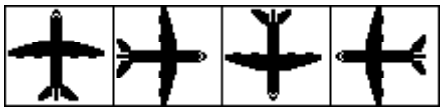
Recall that one of the "next steps" at the end of chapter 4 was to rotate the graphic of the Space Shuttle to show its heading. The need to rotate a screen object, especially one that is animating on the screen, arises frequently in simulations and games. As another example, consider the following:

> You're designing a children's game to help them learn the four directions by having them pretend they are flying an airplane. The goal of the game is to pick up passengers from one location and deliver them safe and sound to another location all the while avoiding hazards such as thunderstorms and other airplanes. As the airplane changes directions (e.g. from North to East), you want the graphic to point in the appropriate direction.

Surprisingly, Authorware does not have a rotate function. The best strategy I have found for achieving this design is creating (or finding) a PICS movie in which the object is rotated a certain number of degrees in each frame. A graphic of the airplane for the game described above would need four frames, each one rotating the airplane 90 degrees:

I created this PICS movie using Macromedia Director (I found the simple airplane graphic in a clip art file and then copied, pasted, and rotated it three times).

frame 1 *frame 2* *frame 3* *frame 4*



A graphic rotated 45 degrees would need eight frames, and so on. The trick is to program Authorware to display only one frame at a time continuously. For example, we would want to only show frame 1 when the airplane is traveling North and only frame 3 when it is traveling South. Fortunately, this is easily done with a variable set to the number of the frame we wish to display.

QuickTime movies and GIF animations work in the same way. Unfortunately, Authorware does not support the import of GIF animations. QuickTime offers the additional functionality of video compression, thus taking up less memory. Although Authorware supports QuickTime movies, these cannot be animated around the screen.

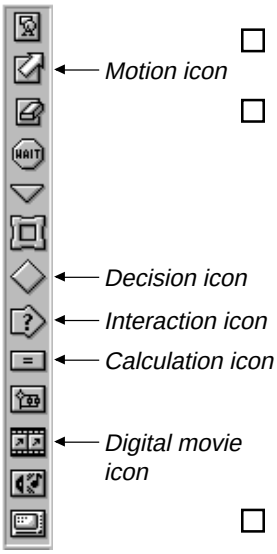
This section briefly demonstrates how to construct such an interactive program. However, it does not show how to create a PICS movie. You will need to already have a suitable PICS movie to complete this section. A sample PICS file is provided in the Rieber Clip Art resource (using the simple airplane graphic shown above).

Getting started

☐ **Launch Authorware and open a new file.**

Let's begin by setting up five user variables. The first, called "frame," will ultimately control the PICS movie itself. As you will see, this variable will tell Authorware which frame of the movie to show. The other four variables will control the motion of the PICS movie around the screen. The variables "h" and "v" will be used inside a Motion icon to determine the position of the movie within a defined screen area. The variables "hspeed" and "vspeed" will be used to control the motion of the movie (from top to bottom, left to right, etc.)

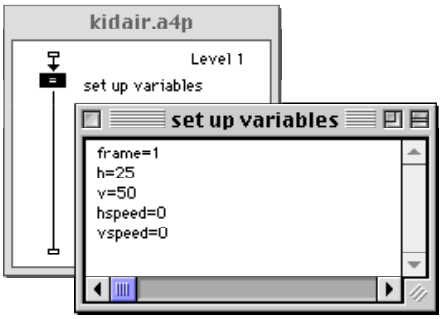
Here are the icons you will be using in this section:



☐ **Drag a Calculation icon to the flow line and title it "set up variables."**

☐ **Double-click to open this Calculation icon.**

☐ **Type the following inside the text window:**



As you'll see, assigning 25 to "h" and 50 to "v" will position the airplane in the left center of the screen.

☐ **When done, click in the close box in the Calculation icon's window.**

☐ **Click "Yes" to save the calculation changes.**

The New Variable dialog box opens.

☐ **For the variable "frame," enter an initial value of 1 and a suitable description, then click "OK."**

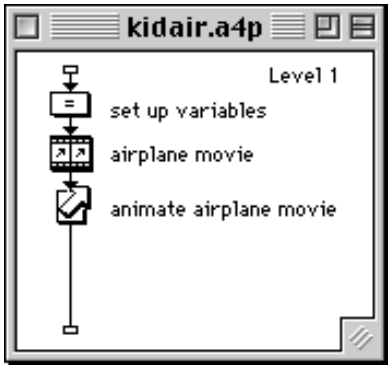
☐ **For the other variables, enter an initial value of 0 and a suitable description, then click "OK."**

Next, let's set up the animation for the airplane. This involves two steps: 1) loading the PICS movie via the Digital Movie icon; and 2) animating this movie around the screen with a Motion icon.

☐ **Drag a Digital Movie icon to the flow line and title it "airplane movie."**

☐ **Drag a Motion icon to the flow line and title it "animate airplane movie."**

Here is what your screen should look like:

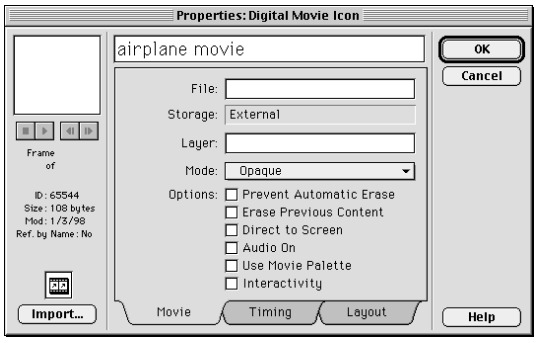


Let's define these icons using the trick of running Authorware and letting it pause the file automatically when it encounters the icons.

You will need to have the PICS movie available on your hard drive in order to perform the rest of the steps in this section.

☐ **Restart the file.**

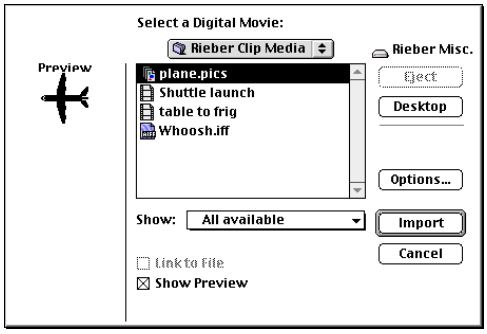
Authorware will pause the file automatically when it encounters the Digital Movie icon "airplane movie," and then open the dialog box for this icon:



Next, we need to import the PICS animation from our hard drive.

☐ **Click on the "Import..." button.**

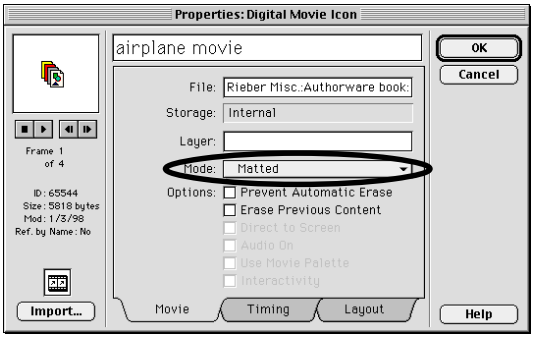
☐ **Find and highlight the PICS movie called "plane.pics," then click "Import":**



The airplane graphic will probably be spinning in the "preview" window if this option is chosen.

Next, you need to make some changes inside different parts of this icon's dialog box.

☐ **While viewing the Movie options, change the "Mode" to "Matted."**



☐ **Click on the Timing tab to view the Timing options.**

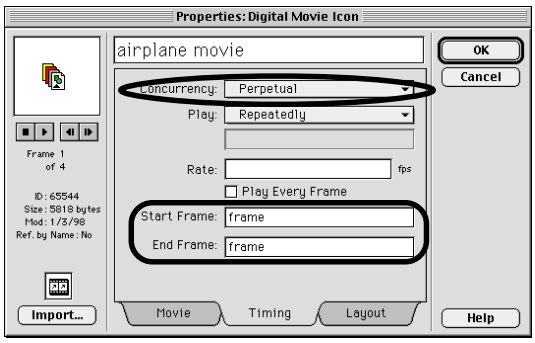
☐ **Insert the variable "frame" inside the two text windows under "Start Frame" and "End Frame."**

Since we are using the same variable in both windows, the movie will play only one of the four frames. In other words, it will start and end with the same frame, holding this single frame on the screen.

☐ **Change the Concurrency to "Perpetual."**

By having it play perpetually, any changes to the variable "frame" will automatically change which frame of the movie is being played (and hence which of the four airplane rotations is displayed).

The dialog box should match the following:



The next step is to work on the animation for this PICS movie. Although the Digital Movie icon has some limited motion options (starting with Version 4.0), we will use a Motion icon to control the animation, just as we have in previous chapters.

☐ **When done, click "OK."**

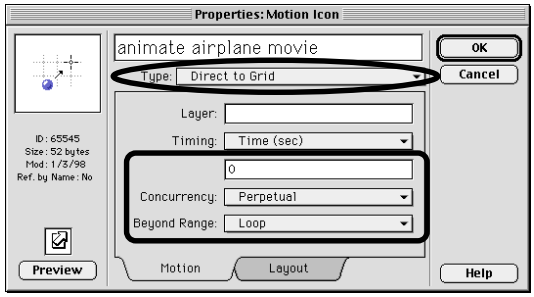
This closes the Digital Movie icon. The next icon on the flow line, the Motion icon "animate airplane movie," automatically opens next.

☐ **Change the motion type to "Direct to Grid".**

☐ **Enter zero (0) in the text box below "Time (sec).**

☐ **Change the Concurrency to "Perpetual."**

☐ **Change the Beyond Range option to "Loop".**



☐ **View the Layout options by clicking on the Layout tab.**

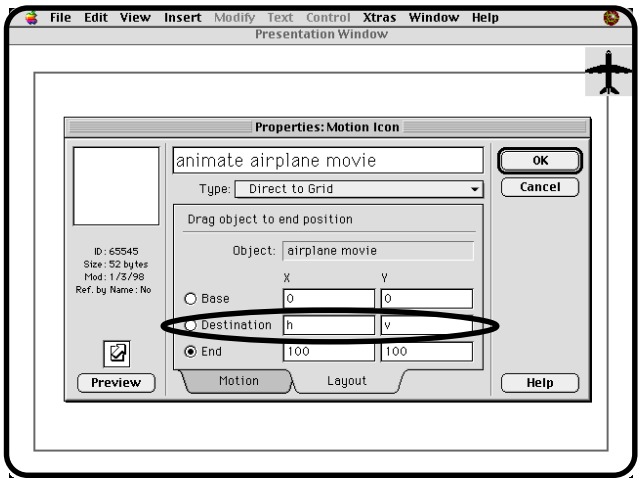
Now we need to define the screen area in which the animation will take place and insert the variables that will control the motion of the object. The instructions that follow are abbreviated given your assumed experience with this procedure from other chapters. Let's start by defining the base and end positions of the animation.

☐ **Make sure that the radio button beside “Base” is checked, then drag the airplane to the bottom left corner of the screen.**

☐ **Check the radio button beside “End,” then move the airplane to the top right hand corner of the screen.**

☐ **Enter the variable “h” in the first variable window beside “Destination” and “v” in the second variable window.**

Your screen should match the following:



☐ **When done, click “OK.”**

☐ **Jump back to the flow line and save the file.**

Setting up the interaction

Next, we need to set up perpetual pushbuttons for each of the four directions that the airplane will travel. Each of these buttons will include a Calculation icon to manipulate the variables that control which frame of the digital movie should be displayed as well as control the motion of the movie around the screen.

☐ **Drag an Interaction icon to the flow line and title it “controls.”**

☐ **Attach a Calculation icon to this Interaction icon.**

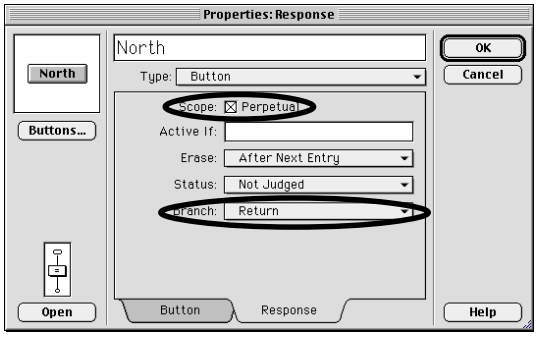
☐ **When the Response Type dialog box appears, be sure the radio button beside “Button” is checked; click “OK.”**

☐ **Title the Calculation icon (or branch) “North.”**

Let’s immediately change this to a perpetual interaction.

☐ **Double-click on the “baby pushbutton” leading into the branch to open the Button Options dialog box.**

☐ **Click on the Response tab to view the response options and turn on the “Perpetual” option and change the branch to “Return”:**

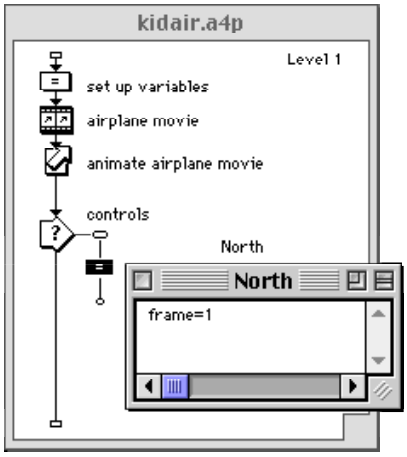


☐ **When done, click “OK.”**

Now let’s modify the variables in this Calculation icon that will control both the movie and the motion of the movie.

☐ **Double-click on the Calculation icon “North.”**

☐ **Type “frame=1” inside the text window:**

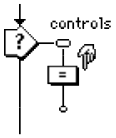


This line sets the variable “frame” to 1. In other words, this line will instruct Authorware to show only the first frame when the user presses this button so that the airplane points North (or up). The other buttons will call for the other frames. We can just copy and paste this button, then make the minor modifications for each one.

☐ **When done, close this Calculation icon's window; click "yes" to save the changes.**

☐ **Click once on the Calculation icon "North" to highlight it.**

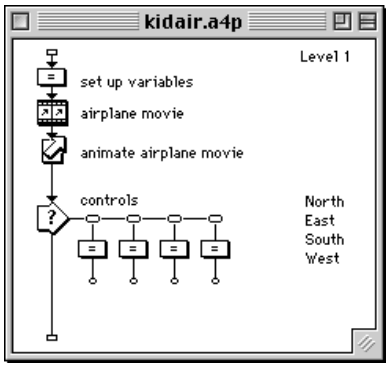
☐ **Select "Copy" from the "Edit" menu.**



☐ **Click once to the right of the Calculation icon "North" so that the paste hand appears on the flow line.**

☐ **Select "Paste" three times from the "Edit Menu."**

☐ **Change the name of these three buttons to "East," "South," and "West" in that order:**



Although the order of the buttons is not important, I chose an order corresponding to the four frames in the digital movie. Now all we have to do is modify the programming inside these three Calculation icons so that the variable "frame" is assigned the value of 2, 3, and 4 respectively.

☐ **Double-click on the Calculation icon "East" and change the programming to "Frame=2", then close this Calculation icon's window and click "yes" to save the changes."**

☐ **Double-click on the Calculation icon "South" and change the programming to "Frame=3", then close this Calculation icon's window and click "yes" to save the changes."**

☐ **Double-click on the Calculation icon "West" and change the programming to "Frame=4", then close this Calculation icon's window and click "yes" to save the changes."**

☐ **Save the file.**

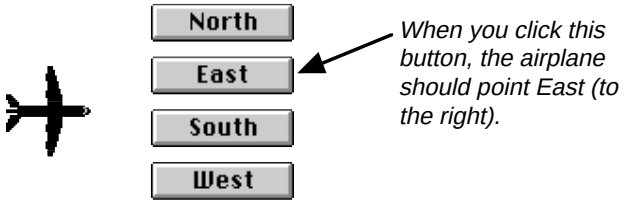
Testing the program

Time for a test to see if the buttons are correctly controlling which frame of the digital movie is displayed.

☐ **Restart the file.**

☐ **Click on each of the four buttons; when you do, the airplane graphic should rotate immediately so that it points in that direction.**

For example, when you click on the button "east," the airplane should immediately change its orientation so that it points to the right:



The reason is that when we click on one of the four buttons, the variable "frame" is assigned a number from 1 to 4 corresponding to that button. As soon as the variable "frame" changes, the Digital Movie icon "airplane movie" immediately changes the starting and ending frame of the movie to match this number since it being played perpetually.

☐ **Jump back to the flow line.**

☐ **Save the file.**

Getting the airplane to fly around the screen

The last thing we will do in this section is finish the animation for the airplane (after all, that is why we included the Motion icon). We will construct a "game engine" similar to what we have done many times in this book. There is no pretense of following any physical laws of motion in this example, however. We just want the airplane to fly at a uniform speed in one of the four directions.

Recall that the Motion icon is using the variables "h" and "v" to control, respectively, the horizontal and vertical positions of the airplane within the screen area we set up earlier. Recall that the position 0,0 is in the bottom left corner of the screen area and the position 100,100 is in the top right corner of the screen area. Therefore, to make the airplane to move from left to right, we need to increase the value of "h." To make it move from right to left, we need to decrease the value of "h." Similarly, we need to increase "v"

to make the airplane go from bottom to top, and decrease "v" to make the airplane move from top to bottom. Fortunately, we set up the variables "hspeed" and "vspeed" for this very reason.

Let's start by setting up the "game engine."

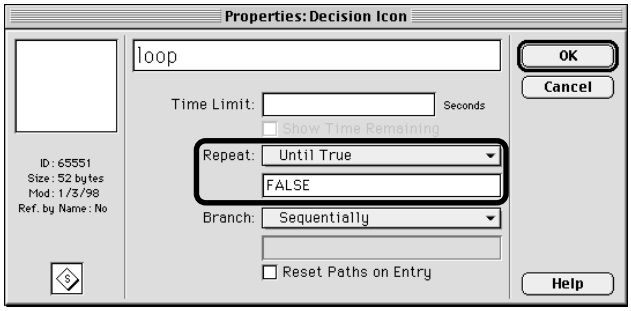
☐ **Drag a Decision icon to the flow line and title it "loop."**

☐ **Attach a Calculation icon to this Decision icon and title it "game engine."**

We need to make this Decision icon loop indefinitely.

☐ **Double-click on the Decision icon "loop."**

☐ **Change the Repeat option to "Until True" and type "FALSE" in the text box below it:**



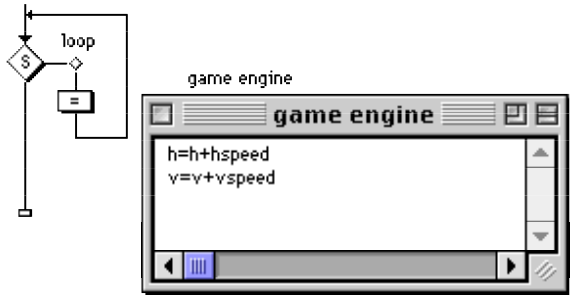
By typing "FALSE" here, the loop will never end. If we were to develop the game completely, we would eventually replace "FALSE" with some condition, such as something that would trigger the end of the game.

☐ **Click "OK."**

Next, let's program the "game engine."

☐ **Double-click on the Calculation icon "game engine."**

☐ **Enter the lines "h=h+hspeed" and "v=v+vspeed" inside the text window:**



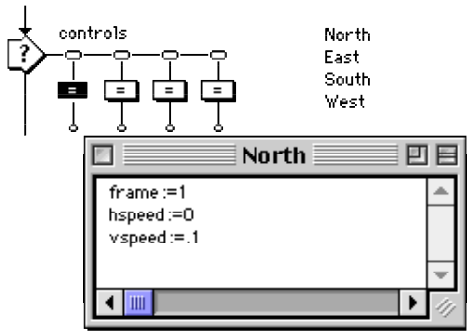
As you can see, as the game engine loops, the current value of "hspeed" and "vspeed" will be added to "h" and "v." As "h" and "v" change, the airplane will move on the screen. Recall that "hspeed" and "vspeed" were initially set to 0. Therefore, "h" and "v" do not change and therefore the airplane does not move until the player clicks one of the buttons.

Of course, we have not yet programmed the buttons to change "hspeed" or "vspeed." Let's do that next.

☐ **Close the Calculation icon "game engine"; click "yes" to save the changes.**

☐ **Double-click on the Calculation icon "North" and add the lines "hspeed=0" and "vspeed=.1" to the text window:**

The value .1 for vspeed is arbitrary. The larger the number, the faster the plane will move. I tried out a few values on my Macintosh PowerPC before settling on this value.



When the player clicks on the button "North," we want the airplane to move due North. By setting "vspeed" to 1 and "hspeed" to 0, the variable "v" will continually increase by 1 in the game engine, whereas "h" will not change (since we are adding 0 to it in the game engine).

We need to use the same logic to program the other three buttons.

☐ **Close this Calculation icon's window; click "yes" to save the changes.**

☐ **Double-click on the Calculation icon "East" and add the lines "hspeed=.1" and "vspeed=0" to the text window.**

☐ **Close this Calculation icon's window; click "yes" to save the changes.**

☐ **Double-click on the Calculation icon "South" and add the lines "hspeed=0" and "vspeed= -. 1" to the text window.**

Careful! *Double-check to make sure you typed this as a negative number (—.1).*

When "vspeed" is a negative number it is the same as subtracting from "v" in the game engine.

☐ **Close this Calculation icon's window; click "yes" to save the changes.**

☐ **Double-click on the Calculation icon "West" and add the lines "hspeed= - .1" and "vspeed=0" to the text window.**

Here too!

☐ **Close this Calculation icon's window; click "yes" to save the changes.**

☐ **Save the file.**

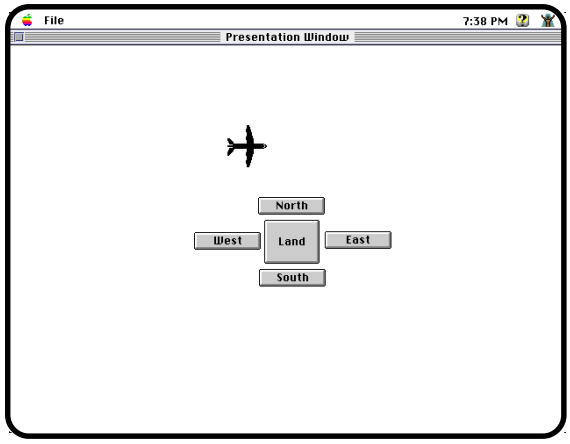
Time for another test

You should now be able to "pilot" the airplane around the screen by clicking the four buttons.

☐ **Restart the file and make the airplane go in different directions by pressing the appropriate buttons.**

If you remembered to choose the option "loop" when setting up the Motion icon, the airplane should "wrap around" the screen when it goes off any edge.

One last thing you should do is improve the user interface by either rearranging the buttons or substituting the buttons for hot spots superimposed on a compass rose. You might also add another button called "land" that sets "hspeed" and "vspeed" to 0:



Technique 3. Creating a "click and hold" response type

Introduction

As you know, Authorware's Interaction icon comes "preprogrammed" to handle a wide range of interactions, such as buttons, hot spots, pull-down menus, etc. However, one response type that seems to be missing is "click and hold" — actions that only take place when the user clicks on a certain screen area and holds the mouse button down. For example, consider the following situations:

> You've developed a series of QuickTime movies that demonstrate how to execute different video camera shots for a course on educational television production. For example, you would like users to get the feeling that they are actually filming a scene in which they have to press and hold the video camera's zoom-in button.
>
> You've developed a series of QuickTime movies that are part of an interactive 3-D adventure taking place in a medieval castle. You used a 3-D modeling application to create the "virtual" castle. This application lets you generate QuickTime movies of real-time "walkthroughs" in and around the 3-D objects. You want the user to get the feeling of being immersed in this virtual world. Your design calls for a simple interface where the user can move forward and backward just by aiming the mouse in the direction of travel, then clicking and holding to "walk" forward or backward through the movie.

There are certainly other ways of accomplishing each of these interactions without the "click and hold" response type, such as creating a series of buttons that control the playing, stopping, rewinding, or stepping through the audio or movie file. If you have played with integrating QuickTime movies in other applications (such as Authorware), you already know that you can usually choose to have the QuickTime controller active to let the user play, pause, and step forward and backward through the movie. However, the "look and feel" of these approaches may not be ideal. For example, consider the second scenario above. You want the user to feel as though they are *in* the castle, not just viewing 3-D images on a computer. There are many instances where the user interface would be improved if a "click and hold" response type were available.

In fact, many 3-D games rely on the "click and hold" response type.

Although this response type is not one of the built-in options of an Interaction icon, we can construct our own using the Conditional response type in tandem with variables and functions.

The goal of this section is to demonstrate the concept, not produce a complete project. To that end, this section will build a short interactive sequence similar to the third scenario described above. But instead of a medieval castle, we will use a QuickTime movie of a short "walk" from the kitchen table to the refrigerator in a 3-D model of my home in Athens, Georgia. This QuickTime movie was generated by Virtus Walkthrough Pro, the 3-D application that I used to model my home on the computer. This QuickTime movie is available in the Rieber Clip Media resources. However, *any* QuickTime movie can be used to complete this section.

A walk with which I'm very familiar!

This programming technique relies on the conditional response type of an Interaction icon in tandem with the following three system variables: MouseDown, CursorX, and CursorY. The variable MouseDown is true if the user is pressing and holding the mouse button (or the left mouse button on Windows computers). The variables CursorX and CursorY contain, respectively, the number of pixels from the left and top edge of the presentation window to the current cursor position. The idea is to define a rectangular screen location and tell Authorware to perform some task if the mouse is both over that area and the mouse button is being held down.

Getting started

☐ **Launch Authorware and open a new file.**

Let's begin by setting up a user variable, called "frame," that will ultimately control the QuickTime movie itself. As you will see, this variable will tell Authorware which frame of the QuickTime movie to play. The QuickTime movie will play forward or backward as the numeric value of this variable increases or decreases.

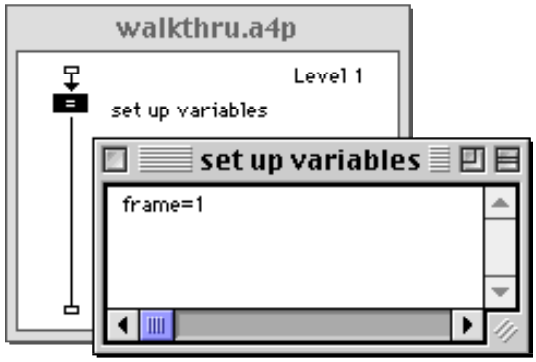
This procedure is almost identical to that used in the previous technique.

A QuickTime movie, like any animation, is a series of single frames that are displayed in succession to produce the illusion of movement. Fortunately, each frame of the movie is addressable permitting us to control which frame is presented and when. The way in which the variable "frame" controls the movie will become clearer when you see this variable in action.

Here are the icons you will be using in this section:

☐ **Drag a Calculation icon to the flow line and title it "set up variables."**

☐ **Double-click on this Calculation icon to open it, then enter the following programming line:**



We set “frame” to 1 because we want the QuickTime movie to start playing at the first frame.

☐ **When done, click in the close box in the Calculation icon’s window.**

☐ **Click “Yes” to save changes to this icon.**

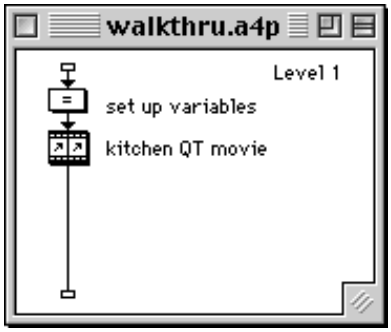
The New Variable dialog box opens.

☐ **Enter an initial value of 1 for this variable and a suitable description:**

☐ **Click “OK.”**

Next, let’s instruct Authorware to load the QuickTime movie we want to control.

☐ **Drag a Digital Movie icon to the flow line and title it “kitchen QT movie.”**

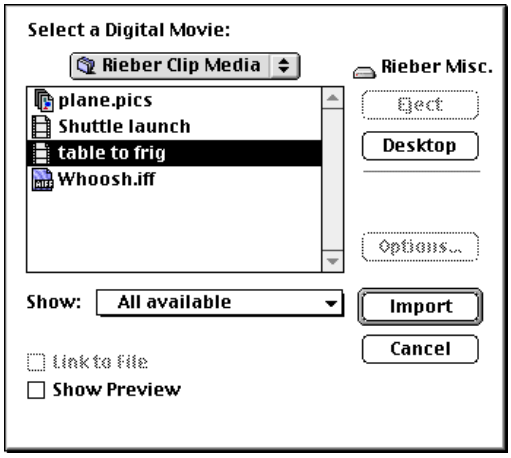


☐ **Double-click on the Digital Movie icon.**

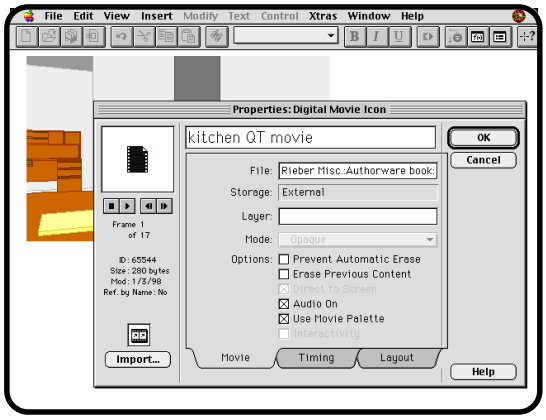
☐ **Click on the “Import...” button.**

☐ **Select the QuickTime movie “table to frig” from the Rieber Clip Media folder, then click “Import”:**

Again, <u>any</u> QuickTime movie will do for the purposes of this example.

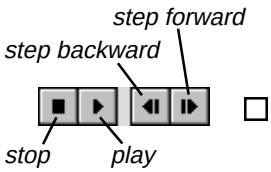


This movie should now appear in the back of the Digital Movie icon's dialog box:



Let's test the movie just to be sure it works.

☐ **Click the play button in the movie controller.**



The movie should begin playing continuously.

There is no audio with this QuickTime movie.

☐ **Click "Stop."**

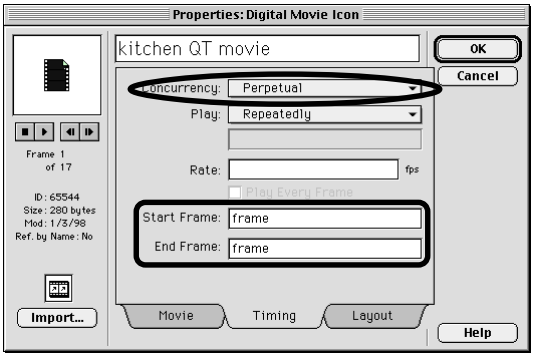
In a moment, we will center the movie on the screen, but first let's go ahead and make two important changes: changing the concurrency of the movie from Concurrent to Perpetual and inserting the variable "frame" in the text boxes beside "Start Frame" and "End Frame."

☐ **Click on the Timing tab to view the Timing options.**

☐ **Change the Concurrency to "Perpetual."**

☐ **Type the variable "frame" in the text boxes beside "Start Frame" and "End Frame" with the variable "frame."**

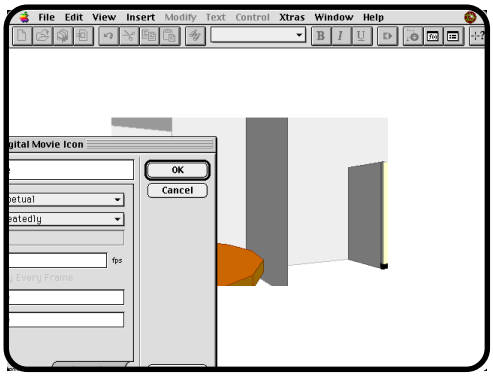
When finished, your dialog box should match the following:



In this way, the variable "frame" will control the movie. As "frame" increases, the movie will play forward. When it decreases, the movie will play backwards. When it stays the same, it will play the same frame over and over (giving the illusion that the movie has paused).

Finally, let's center the movie on the screen.

☐ **Center the digital movie on the screen by clicking, holding, and dragging it to the new location:**



You may need to move the dialog box out of the way to do this properly.

☐ **When done, click "OK."**

☐ **Save the file.**

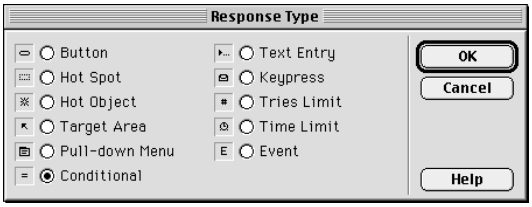
Remember to save periodically throughout this session.

Setting up the user interaction

It's time to build the interactive features of the program. As always, we will use the Interaction icon to accomplish this. However, since there is no "click and hold" response type we need to program our own. This can be done with the conditional response type.

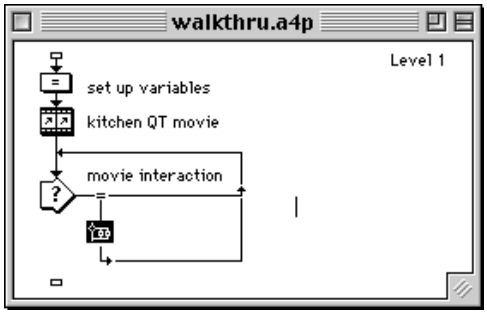
☐ **Drag an Interaction icon to the flow line and title it "movie interaction."**

☐ **Attach a Map icon to this Interaction icon. When the Response Type dialog box opens, select "Conditional":**



☐ **Click "OK."**

You should now be back at the flow line, with the cursor flashing to the right waiting for you to give this conditional branch a title:



However, the title we choose for this branch must be written carefully because it acts as the *programming* for the branch.

Let's recall what we want, described in human terms. When the user clicks and holds on the movie appearing on the screen, we want the movie to start playing. If the user releases the mouse button, the movie stops. The user can thus play/pause the movie in this way. The programming for the feature requires two elements: 1) a way to tell if the user is holding down the mouse button; and 2) a way to tell if the mouse is currently over the screen area covered by the movie.

The first element can easily be achieved via the general system variable "MouseDown" — this variable simply returns a value of TRUE when the user is pressing the mouse button and FALSE when the user is not. The second element can be achieved via the system variables "CursorX" and "CursorY." These two variables are constantly keeping track of the screen coordinates of the mouse. CursorX is the horizontal position of the cursor (measured as the distance from the upper left hand corner of the screen) and CursorY is the vertical position of the cursor (measured by the distance from the upper left hand corner of the screen).

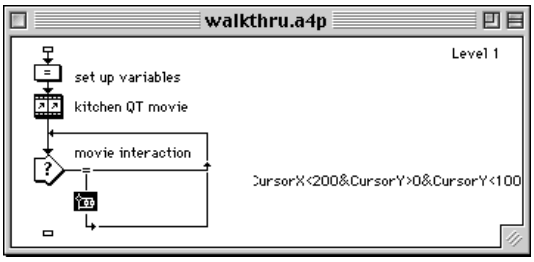
Let's illustrate this with an example.

☐ Type the following title for the conditional branch:

Type carefully!

MouseDown&CursorX>0&CursorX<200&CursorY>0&CursorY<100

The ampersand (&) signifies "and" in the logic of this line.



This line is too long for all to stay on the screen at the same time. Remember that you can use the keyboard arrows to navigate left and right.

Realize that we chose these coordinates for convenience — the movie graphic does not exactly cover that area. We'll modify these coordinates after we verify that the programming works.

This line instructs Authorware, in human terms, to execute this branch *if* the mouse button is down and the cursor is between 0 and 200 horizontally and the cursor is between 0 and 100 vertically.

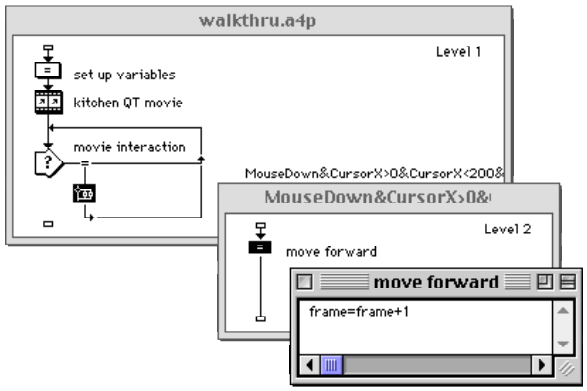
You will understand this more after you run the file and see it in action. But before we can test it, we need to embed instructions for making the movie play one frame at a time inside this Map icon.

☐ **Double-click to open the Map icon of the conditional branch.**

☐ **Drag a Calculation icon to this Level 2 flow line and title it "move forward."**

☐ **Double-click to open this Calculation icon.**

☐ **Type "frame=frame+1" in the text window:**



This increases the variable "frame" by one. Recall that this is the variable that we inserted into the Digital Movie icon to control the start and end frame displays. As "frame" increases the movie will play forward.

☐ **When finished, click in the close box in the Calculation icon's window.**

☐ **Click "Yes" to save the calculation changes.**

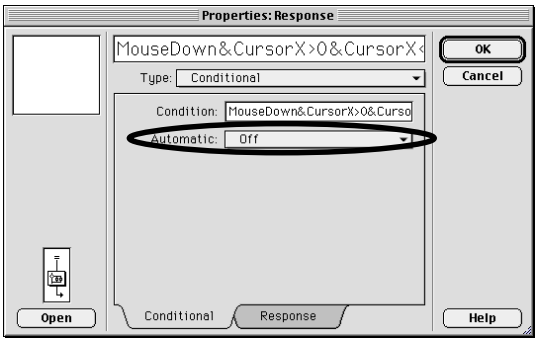
There is one more thing we have to do before this branch will work. We need to turn on the "Automatic" matching feature of this conditional branch.

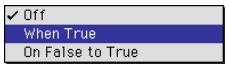
This ensures that the branch will be triggered at the moment the condition is met and is not dependent on the flow line.

☐ **Open the Conditional Options dialog box by double-clicking on the "baby equal sign" just above the Map icon leading into the conditional branch.**

☐ **Be sure you are viewing the Conditional options.**

Notice that the “Automatic” matching option is currently off:



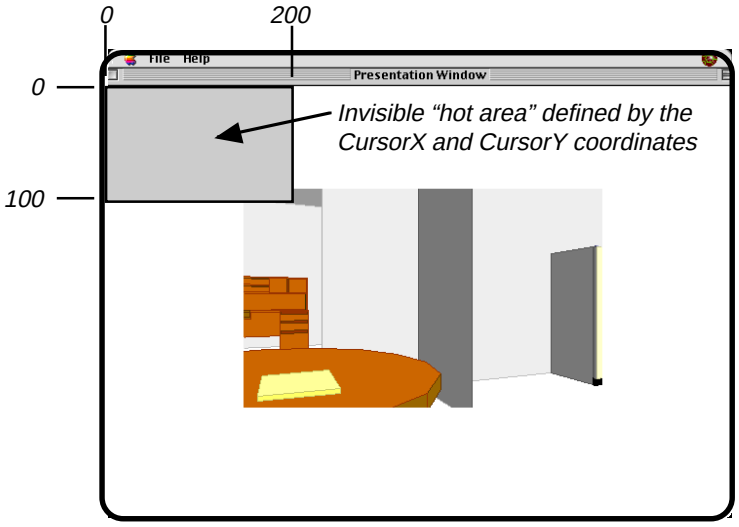


☐ **Change the Automatic matching option to “When True.”**

☐ **Click “OK.”**

Testing the response branch

Time to test this “click and hold” response branch that we have just programmed. Recall that we set the horizontal screen coordinates (with the system variable “CursorX”) to an area between 0 and 200 and the vertical screen coordinates (with the system variable “CursorY”) to an area between 0 and 100. Therefore, the screen area that is “hot” will be an area in the top left hand corner of the screen:



We will soon modify this so that this “hot area” exactly matches the screen area occupied by the movie.

Let’s run the program to test it.

☐ **Restart the file.**

☐ **Click and hold in the top left hand corner of the screen to advance the digital movie one frame at a time.**

It's easy to accidentally double-click when testing this. You may inadvertently open whatever icon is associated with the object beneath the cursor. If this happens, just click "OK" to move on.

The movie should not play unless you click and hold in the "hot area" and the mouse button is held down.

☐ **Jump back to the flow line.**

Modifying the screen coordinates to match the movie

We need to move and resize the "hot area" so that it matches the screen area in which the digital movie is located. We could use a trial and error method to find these coordinates, but a better way is to use the system variables "CursorX" and "CursorY" to identify the exact coordinates quickly. To do this, we need to embed these variables in the display and have Authorware to update their values continuously.

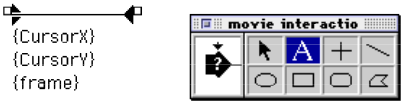
☐ **Double-click on the Interaction icon "movie interaction."**

Let's embed the following three variables: CursorX, CursorY, and Frame.

☐ **With the text tool, type each of the three variables within a set of braces near the top center of the screen:**

Remember that you can also choose to have Authorware paste these variables in for you.

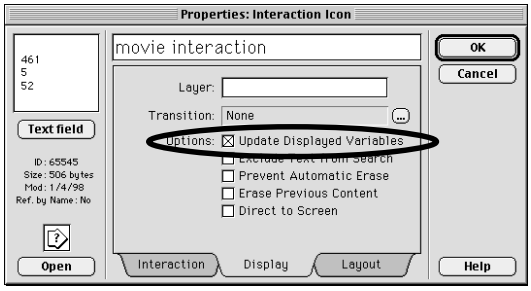
These variables will be displayed only temporarily. When finished, remember to delete this text window from the display.



Next, we need to have Authorware automatically update these variables while the program is running.

☐ **Select "Icon" from the "Modify" menu, then select "Properties...".**

☐ **Click on the Display tab to view the Display options, then turn on "Update Displayed Variables" option:**



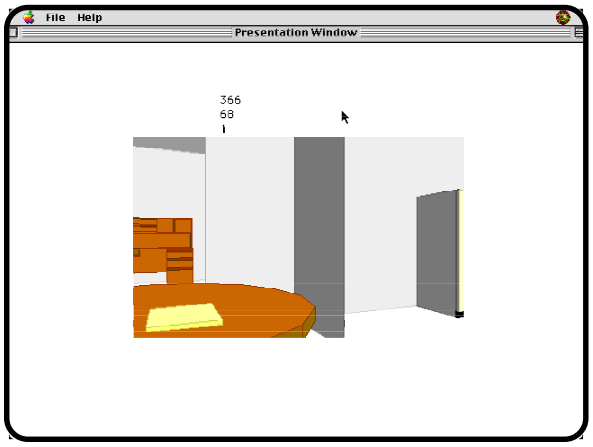
☐ **Click "OK."**

☐ **When finished, click in the close box in the tool box to go back to the flow line.**

Let's run the file to see how this works.

☐ **Restart the file.**

As you move the cursor around the screen, notice how the first two numbers change continuously. These are showing the exact horizontal and vertical coordinates of the mouse on the screen:



The third number remains a "1" since this is the current value of "frame." If you click and hold in the current "hot area" you will see the value of "frame" change as the movie plays. Of course, the movie only has 18 frames, so it stops playing when "frame" exceeds this number.

Let's find the exact screen coordinates for the four edges of the digital movie.

☐ **Move the mouse so that the tip of the pointer aligns perfectly with the top left corner of the digital movie; write down the screen coordinates (the first two numbers) on a piece of paper.**

An error of a couple of pixels will not matter.

In my case, the left edge is at 140 (the first number) and top edge is at 105 (the second number).

☐ **Move the mouse so that the tip of the pointer aligns perfectly with the bottom right corner of the digital movie; write down the screen coordinates (the first two numbers) on a piece of paper.**

In my case, the right edge is at 508 (the first number) and bottom edge is at 329 (the second number).

The next step is to modify the title of the conditional response branch to include these new coordinates.

☐ **Jump back to the flow line.**

☐ **Modify the title of the conditional response branch by entering the new coordinates that match the screen location of the digital movie.**

Again, remember you can use the keyboard arrows to navigate your position on this line in order to make the necessary changes.

In my case, the modification would look like this:

MouseDown&CursorX>140&CursorX<508&CursorY>105&CursorY<329

left edge	*right edge*	*top edge*	*bottom edge*

☐ **Restart the file to test the new coordinates.**

The movie should only now play when the mouse is held down directly over the movie.

☐ **Jump back to the flow line.**

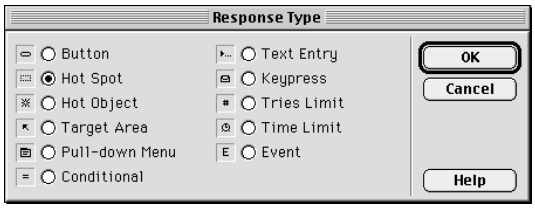
☐ **Save the file.**

Improving the user interface

An improvement to this design would be to have the mouse change shape from the pointer (arrowhead) into one of the custom cursor shapes (such as the hand). Unfortunately, there are no options for changing the cursor shape within the conditional response type as there are for the "hot spot" response type. No problem. We can just set up another branch using the "hot spot" response type and take advantage of this option even though this branch will serve no other function.

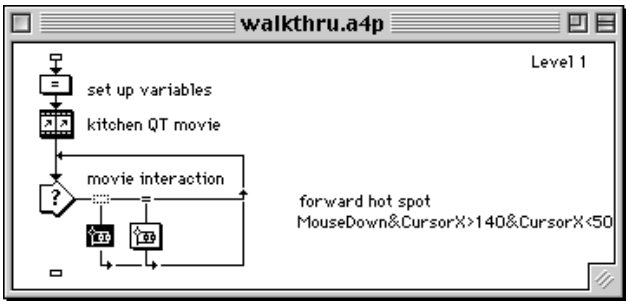
☐ **At the flow line drag another Map icon to the flow line and attach it between the Interaction icon and the conditional response branch.**

☐ **Select the "hot spot" response type, then click "OK":**



Click and hold response type

☐ **Title this branch “forward hot spot”:**



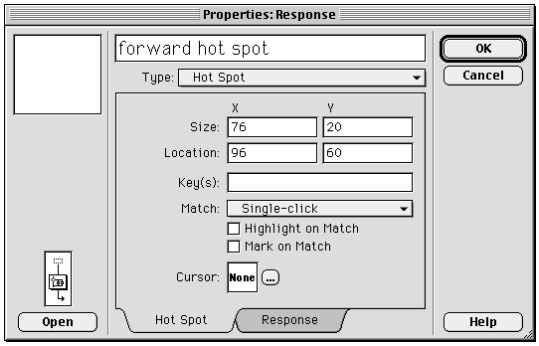
We need to reposition the hot spot over the screen area occupied by the digital movie, then change to the custom cursor shape. We’ll do both tasks by running and pausing the file.

☐ **Restart the file.**

☐ **Press Command-P to pause the file.**

The hot spot associated with this branch appears on the screen.

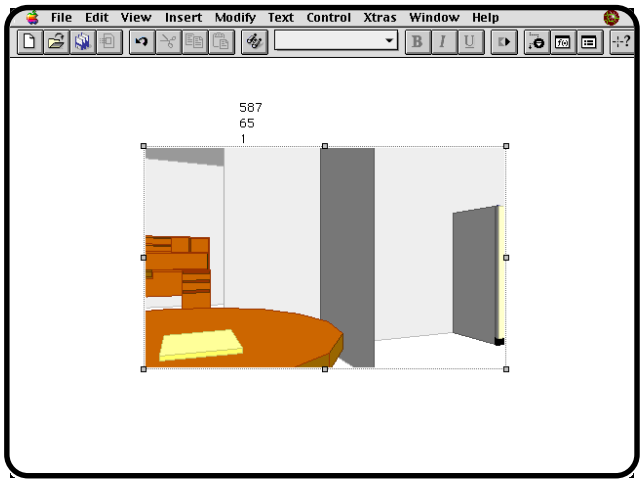
☐ **Double-click on the hot spot to open its options dialog box:**



☐ **Be sure you are viewing the “Hot Spot” options, then change the Cursor from “None” to the hand symbol.**

☐ **Click “OK.”**

☐ **Move and resize the hot spot so that it covers the entire screen area occupied by the digital movie:**



☐ **Press Command-P to proceed.**

As you move the mouse around the screen, the cursor will change from the pointer to the hand when it passes over the movie. This will provide an unobtrusive clue that this is a "hot area."

Reminder! *This additional branch performs no other function. We are using it solely to take advantage of the custom cursor option available within the "hot spot" response type.*

☐ **Be sure that the program functions properly.**

☐ **Jump back to the flow line.**

☐ **Save the file.**

This concludes the step-by-step demonstration on how to build the "click and hold" response type.

Next steps

There are several enhancements that you could make at this point. For example, as you stepped forward through the movie, you may have felt the desire to step *backward*. This is easily accomplished by first displaying a "Go Back" graphic on the screen (such as via the Interaction icon), then copying the pasting the conditional response branch, changing the screen coordinates to match the "Go Back" graphic, and finally by changing the line of code in the Calculation icon to "frame=frame-1."

You will also need to program Authorware not to let the variable "frame" increase to a number larger than the number of frames in the movie (18 in the above example) when moving forward or decrease to a number smaller than 0 when going backward.

To do this, add this line to the Calculation icon in the conditional branch that steps the user forward:

```
if frame>18 then frame=18
```

Then, add this line to the Calculation icon in the conditional branch that steps the user backward:

```
if frame<0 then frame=0
```

Of course, it would be best if the "Go back" graphic does not appear *until* the user has made their first steps forward since technically the user cannot go backward when at frame 1.

Another extension of this technique would be to give the user the option to go left and right. For example, this movie sequence shows another room off to the back left of the breakfast room. The trick here is to build different routes leading to different QuickTime movies via the Decision icon or Framework icon.

Technique 4. Building a response palette that users can move

Introduction

The inspiration for this technique comes from the work of Kelly Tribble, a doctoral candidate at The University of Georgia.

You are already well acquainted with the idea of movable palettes. They are a common design within most applications using a graphical user interface (GUI). For example, consider the many palettes you have been using in Authorware when a Display icon is opened. The tool box, fill palette, color palette, line palette, and modes palette are all examples. You know how cluttered the screen can become when too many palettes are open. You also know how important it is to be able to move these palettes on the screen while you are working. Likewise, our screens become more cluttered and users become more frustrated as we add more and more hot spots and screen buttons to our games and simulations. Consider the following situations:

The idea for such an interactive adventure for learning about history belongs to David Noah, another doctoral candidate at The University of Georgia.

> You're developing a highly interactive simulation/game in history. In it, you present the user with a variety of virtual environments modeled after places in Abraham Lincoln's life, such as the log cabin in which he was born in Kentucky, his law office in Springfield, Illinois, and the White House in Washington, DC. In it, the user can roam around any of these places, picking up and interacting with various objects to learn about them and their relationships to Lincoln's destiny. However, you are finding it necessary to give the user many response palettes, such as general controls for moving between the virtual environments, specific controls for moving within any one place, an interactive time line, a dictionary of names, places, and events, and so on. You want to give users the control to place these response palettes anywhere they wish on the screen as well as closing palettes that they aren't currently using (with the option to open these again at will via a pull-down menu).
>
> You've greatly enhanced the simple version of the Space Shuttle Commander simulation built in chapter 4. You now have several space stations as targets and these are all animating slowly on the screen. You want to change the "shuttle control console" into a movable response palette because you are finding that it often covers critical areas of the screen.

Again, the goal of this section is only to demonstrate the concept, not produce a complete project. Therefore, this section will show how to build a simple response palette con-

sisting of two choices — "True" and "False" that can be moved anywhere on the screen.

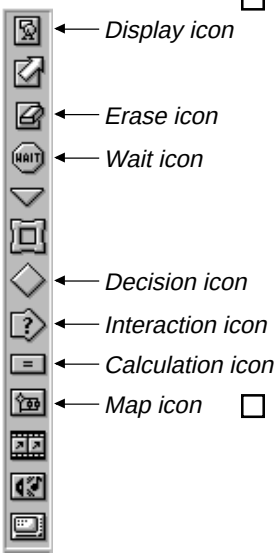
This section has many similarities to the previous section on building a "click and hold" response type. You are encouraged to complete that section first.

Getting started

☐ **Launch Authorware and open a new file.**

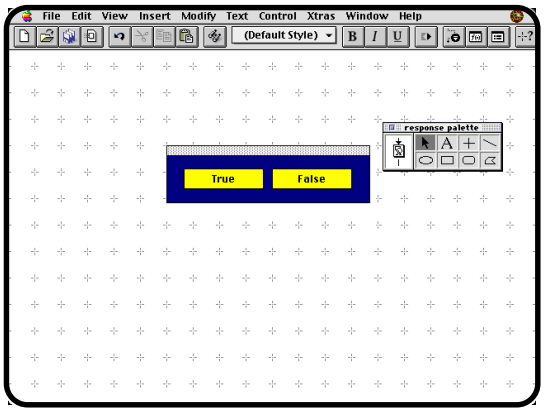
Let's begin by drawing the display for the response palette itself. This needs to set up in its own Display icon.

Here are the icons you will be using in this section:



☐ **Drag a Display icon to the flow line and title it "response palette:"**

☐ **Double-click to open this Display icon and build a simple response palette in the center of the screen with the rectangle tool and text tool similar to the following:**



Notice my attempt to give this response palette a handle bar on the top. However, when done, the user will be able to grab any part of the response palette to move it. Even so, using this graphic design helps to convey to the user that this is a movable object.

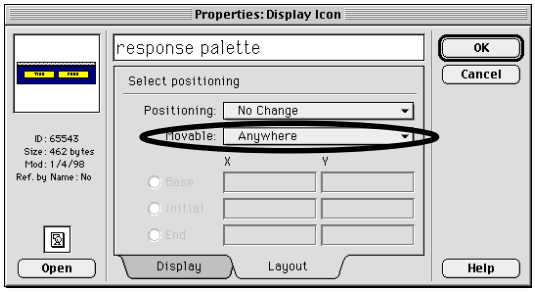
We will be building hot spots in a few minutes that will cover the screen areas behind the words "True" and "False."

We also need to tell Authorware to allow the user to freely move this display on the screen.

☐ **Select "Icon" from the "Modify" menu, then select "Properties...".**

☐ **Click on the Layout tab to view the Layout options.**

☐ **In the dialog box, change the "Movable" option to "Anywhere"; then click "OK."**



☐ **When done, click in the close box in the tool box to go back to the flow line.**

☐ **Save the file.**

Remember to save periodically throughout this session.

Setting up variables

This technique relies heavily on two variables that keep track of the current screen position of the response palette display that we just constructed: one for the position of the left edge of the response palette and one for its top edge. The position of the two hot spots will then be set relative to these two positions.

☐ **Drag a Calculation icon to the flow line and title it "response palette variables."**

☐ **Double-click to open this Calculation icon.**

☐ **Type the following inside the text window:**

You could also choose to have Authorware paste these system variables in for you. They are located in the "Icon" category of variables.

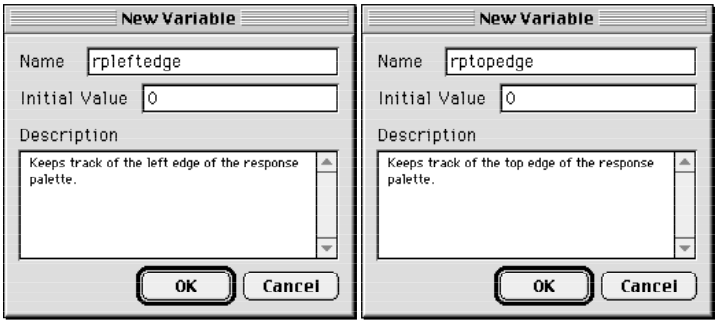
"DisplayLeft" and "DisplayTop" are system variables that enable you to pinpoint the location of a display or digital movie in the icon specified. As you can see, we are passing this information off to two of our own variables — "rpleftedge" and "rptopedge." These will consequently keep track of the location of the left and top edges of the response palette.

☐ **When done, click in the close box in the Calculation icon's window.**

☐ **Click "Yes" to save the calculation changes.**

The New Variable dialog box opens.

☐ **For both variables, enter an initial value of 0 and a suitable description (such as the following), then click "OK":**



You will see how these two new variables to work as we create the two hot spots.

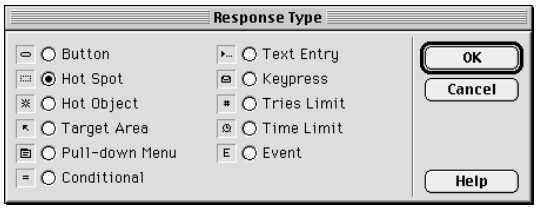
Creating a hot spot with a relative position

We need to create two hot spots, one for "True" and one for "False." The unique aspects of these two hot spots is that they "follow" the response palette as it moves on the screen. However, we begin as we would to create any interaction by dragging an Interaction icon to the flow line.

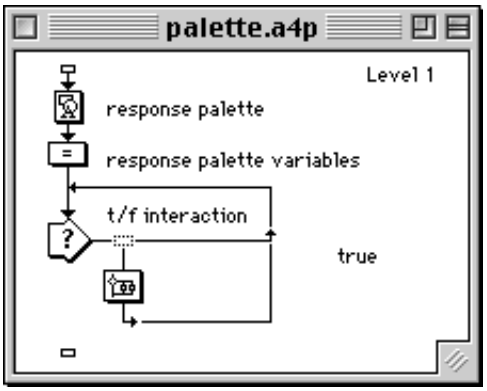
☐ **Drag an Interaction icon to the flow line and title it "t/f interaction."**

☐ **Drag a Map icon to the flow line and attach it to the Interaction icon.**

☐ **Select "hot spot" from the response type dialog box, then click "OK."**



☐ **Title this branch "true."**

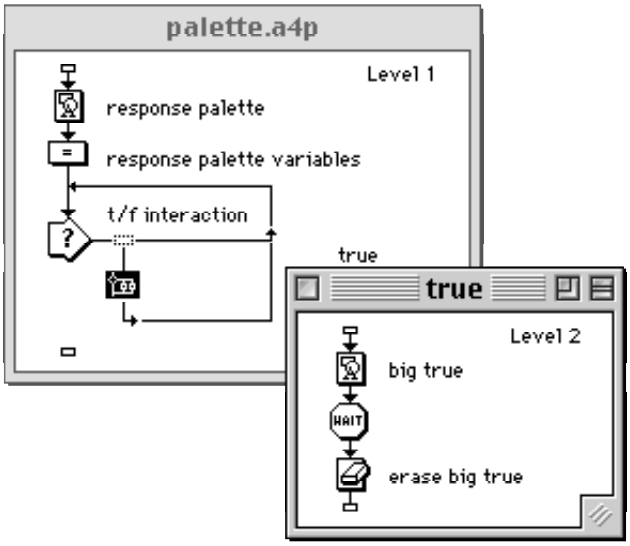


Let's go ahead and build a simple action to take place when this hot spot is clicked. This way, we will know if the hot spot is working. This action can obviously be replaced with any action needed by your game or simulation. When clicked, let's have this branch display the word "True" in big letters at the top of the screen for 1 second.

☐ **Double-click on the Map icon "true."**

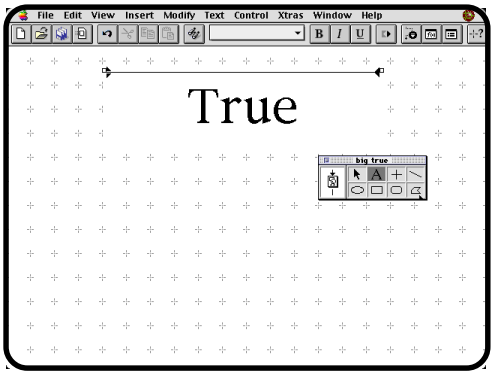
☐ **Drag a Display icon, Wait icon, and Erase icon to this level 2 flow line.**

☐ **Title them as per the following:**



☐ **Double-click to open the Display icon "big true."**

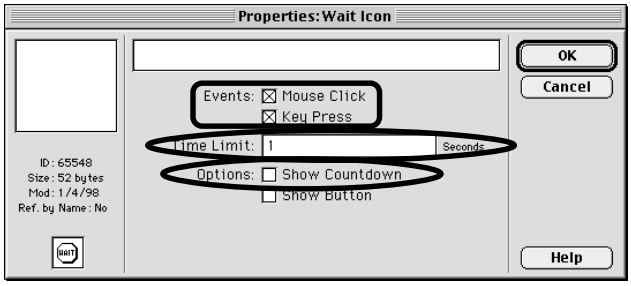
☐ **With the text tool, type the word “True” in big letters at the top of the screen:**



☐ **When done, click in the close box in the tool box to go back to the flow line.**

☐ **Double-click on the Wait icon.**

☐ **Set the time limit to 1 second; select “Mouse Click” and “Keypress”; do not select “Show Button”:**



☐ **Click “OK.”**

☐ **Double-click on the Erase icon “erase big true”; choose to erase the display associated with the Display icon “big true”; click “OK.”**

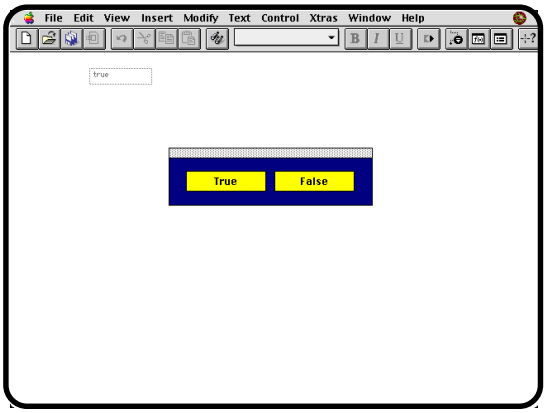
Now, let’s run the file to continue setting up the hot spots.

☐ **Restart the file.**

The response palette appears. Let’s pause the file to fine the invisible hot spot.

☐ **Press “Command-P” to pause the file.**

true



☐ **Move and resize the hot spot so that it fits exactly over the display of the "True" button.**

Let's test the hot spot just to make sure it works.

☐ **Press "Command-P" to proceed.**

☐ **Click once on "True" inside the response palette.**

The word "True" in big letters should on top of the screen for one second.

☐ **Jump back to the flow line.**

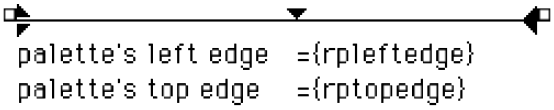
Before we can start moving the response palette around the screen, we need to set up two more variables for the hot spot we just created. These next two variables will tell Authorware where to put the left and top edge of the hot spot. The idea is to set the hot spot to a fixed position *relative* to the response palette. To do this, we will use the trick of temporarily displaying the variables "rpleftedge" and "rptopedge" on the screen along with the current location of the cursor (via the system variables "CursorX" and "CursorY").

This was the same trick used in the previous section that showed how to build a "click and hold" response type.

We will temporarily display these variables with the use of the Interaction icon "t/f interaction."

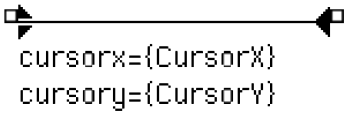
☐ **Double-click to open the Interaction icon "t/f interaction".**

☐ **With the text tool, type the following toward the bottom left of the screen:**

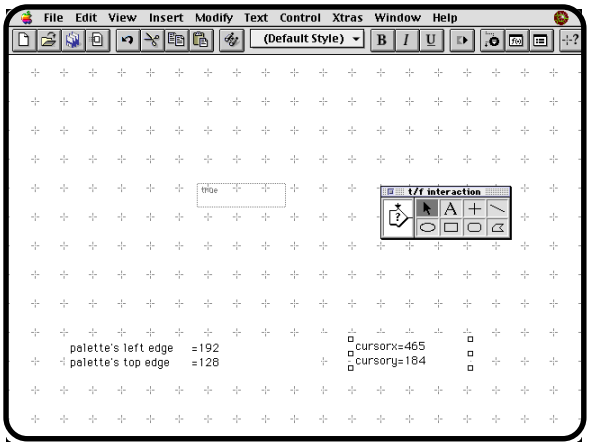


The small triangle above the text-width line is a tab stop produced by clicking anywhere on the line. It can be dragged anywhere on the line.

☐ **With the text tool, type the following toward the bottom right side of the screen:**



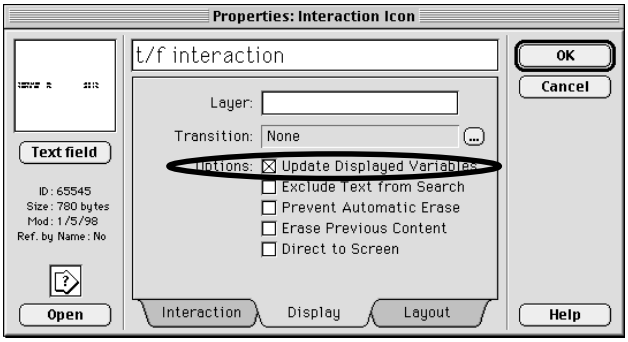
When done, you will have information about the location of the palette and cursor displayed on the screen:



☐ **Select ""Icon" from the "Modify" menu, then select "Properties...".**

☐ **Click on the Display tab to view the Display options.**

☐ **Turn on "Update Displayed Variables":**



☐ **Click "OK."**

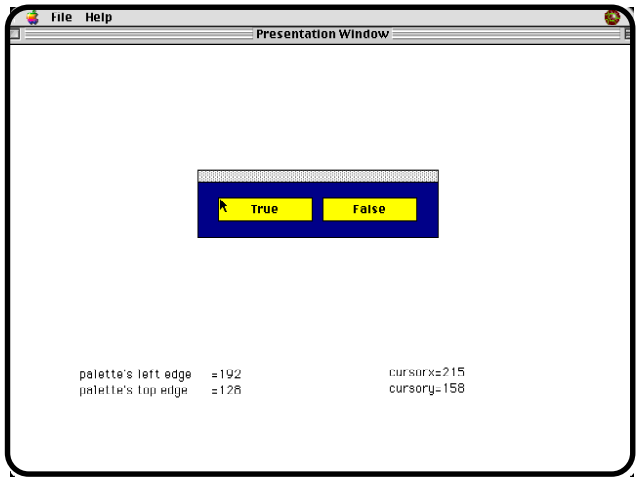
☐ **When done, click in the close box in the tool box to jump back to the flow line.**

We are finally ready to gather some data about the relative distance of the hot spot from the left and top edge of the response palette display.

☐ **Restart the file.**

Displayed on the screen is the location of the response palette and the continually changing position of the cursor (the pointer). By placing the cursor in the top left hand corner of the "True" button display, we can determine the relative position of the hot spot.

☐ **Align the mouse cursor so that it points directly to the top left hand corner of the "True" button display:**



☐ **Subtract the palette's left edge from the CursorX position to determine the relative position of the hot spot's left edge.**

In my case, this value comes to 23 (i.e. 215 minus 192).

☐ **Subtract the palette's top from the CursorY position edge to determine the relative position of the hot spot's top edge.**

While you're here, you might as well make the same measurement to determine the relative position for the other to-be-developed hot spot that will cover the "false" display button.

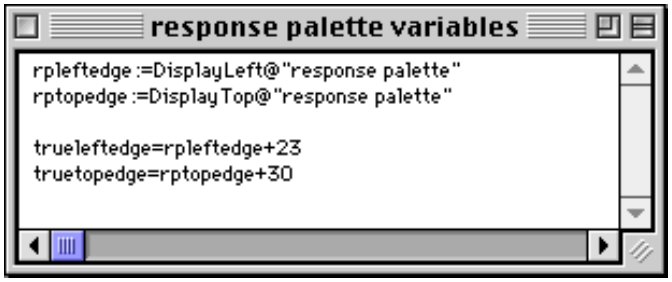
In my case, this value comes to 30 (i.e. 158 minus 128).

With this information we are ready to set up two new variables that will control the relative position of this hot spot.

☐ **Jump back to the flow line.**

☐ **Double-click to open the Calculation icon "response palette variables."**

☐ **Insert the following two lines using the relative position values you calculated in the previous step:**



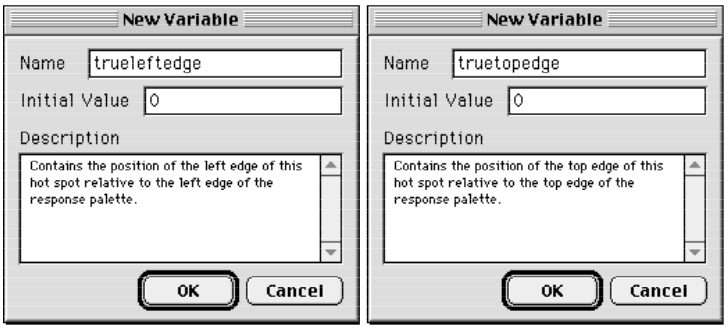
As you can see, we are creating two new variables that will contain the position of the left and top edges of the hot spot relative to the response palette.

☐ **When done, click in the close box in the Calculation icon's window.**

☐ **Click "Yes" to save the calculation changes.**

The New Variable dialog box opens.

☐ **For both variables, enter an initial value of 0 and a suitable description (such as the following), then click "OK":**



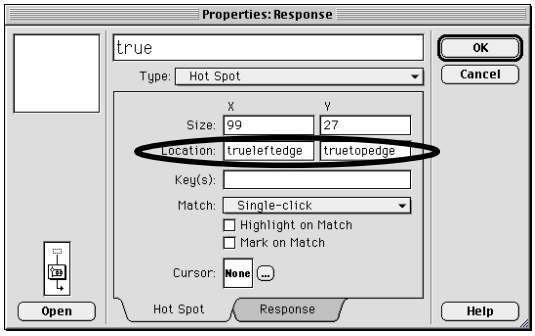
Next we need to tell Authorware how to use these two new variables to move the hot spot to this relative position.

☐ **Open the Hot Spot Options dialog box by double-clicking on the "baby hot spot" leading into the branch titled "true."**

☐ **Be sure you are viewing the Hot Spot options.**

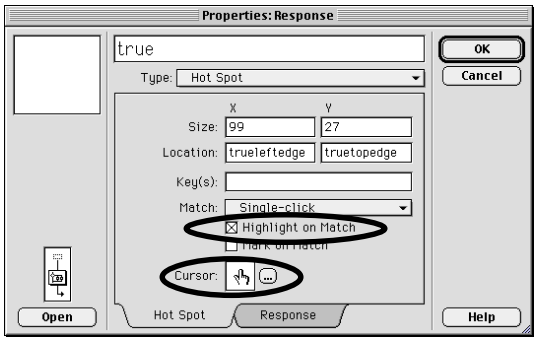
This dialog box shows the absolute location and size of the hot spot. We will leave the hot spot's size alone since it does not change. However, we need to replace the location values with the variables "trueleftedge" and "truetopedge."

☐ **Enter the variables "trueleftedge" and "truetopedge" in the appropriate text windows:**

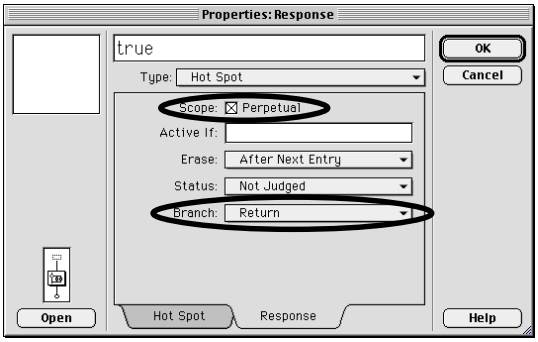


This hot spot needs a few more modifications, the most important of which is that it need to be a perpetual interaction. We also might as well improve the user interface by changing to a custom cursor and having this hot spot "Auto Highlight" when clicked. So, before we leave the Hot Spot Options dialog box, let's make these changes.

☐ **While viewing the Hot Spot options change the custom cursor from "None" to the hand symbol and turn on the option "Highlight on Match":**



☐ **Click on the Response tab to view the Response options; turn on the "Perpetual" option and change the Branch to "Return":**



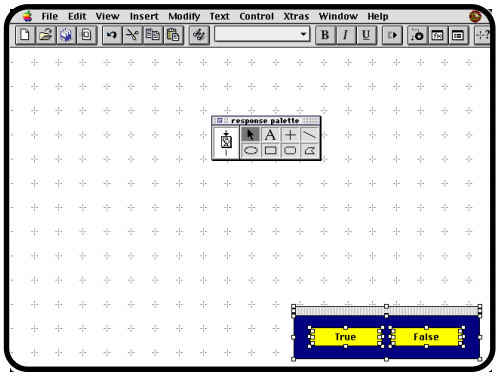
☐ **When done, click "OK."**

Testing the program

Unfortunately, we are not ready to test the program yet in the way we would like, that is, by moving the response palette to a variety of new locations *while the program is running*. We still have some work to do before that test is possible. However, we can perform a simple test at this point just to ensure that the programming logic we've constructed so far is working properly.

☐ **At the flow line, double-click to open the Display icon "response palette."**

☐ **Be sure that all of the screen objects are selected** (such as by pressing "Command/Control-A")**, then move the entire response palette to a new screen location** (such as the bottom right corner of the screen)**:**



☐ **When done, click in the close box in the tool box to return to the flow line.**

☐ **Restart the file and click on the true display button to test whether the hot spot is really there.**

☐ **Repeat the previous four steps several times, each time moving the response palette to a new screen location, to check that the programming logic is working properly.**

☐ **Jump back to the flow line.**

☐ **Save the file.**

You've completed the programming for this hot spot. You will need to eventually repeat these procedures to create a hot spot for the "False" display button. However, we are going to tackle next the remaining programming necessary to make this a fully functioning movable palette.

You'll probably see the text display flicker on the screen when you run the file. This is due to our changing of the hot spot to Perpetual. The Interaction displays the text, the quickly erases as the flow exits the icon. If this bothers you, just double-click on the Interaction icon and change the "Erase interaction" option to "Don't erase."

Updating the variables when the user moves the palette

When we run the program, Authorware checks the current position of the response palette via the Calculation icon "response palette variables" and then aligns the hot spot to its relative position on the screen. Even though the user can now freely move the response palette anywhere on the screen, we still have not programmed Authorware to update the position of the hot spot after each move. We will accomplish this with another perpetual branch attached to the Interaction icon "t/f interaction." However, we will use a special response type called a "target area." This response type allows us to tell Authorware to perform an action when it notices if a particular screen object is moved. We will embed a copy of the Calculation icon "response palette variables" inside this "target area" branch.

☐ **Drag a Map icon to the flow line and attach it to the Interaction icon on the far right side.**

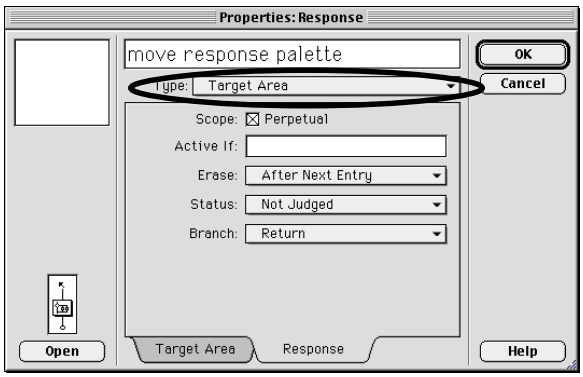
It is very important that this branch be last within the Interaction group. The programming won't work otherwise.

☐ **Title this Map icon "move response palette."**

We need to change the response type for this branch from a hot spot to a target area.

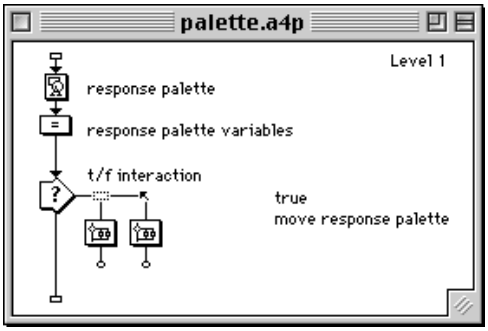
☐ **Double-click on the "baby hot spot" just above the Map icon "move response palette" to open the Hot Spot Options dialog box:**

☐ **Change the response type to "Target Area":**



☐ **Click "OK."**

This sends you back to the flow line:



Next, we have to program Authorware to detect if the user has moved the response palette anywhere on the screen. We can do this just by running and pausing the file.

☐ **Restart the file.**

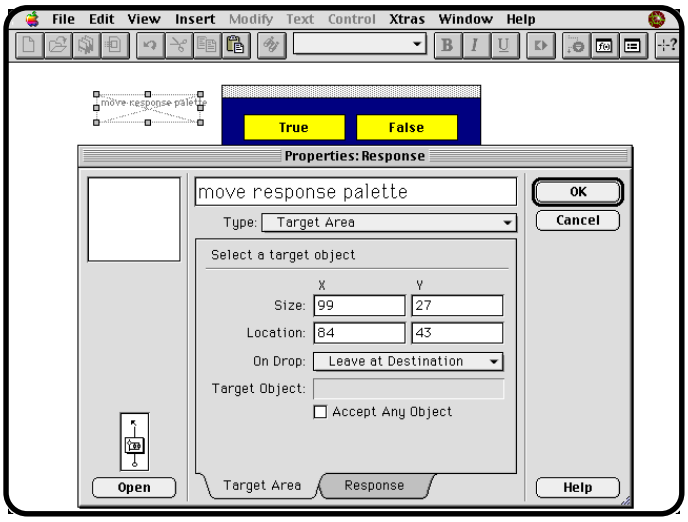
☐ **Press "Command-P" to pause.**

You'll see an outlined rectangle with an "X" in it on the screen. This is the target area.

☐ **Double-click on an edge of the target area.**

This reopens the Target Area Options dialog box.

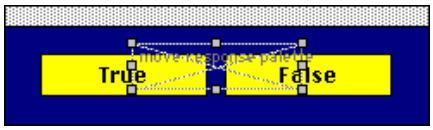
☐ **Click on the Target Area tab to view the Target Area options:**



We will follow the directions in the dialog box: "Select a target object."

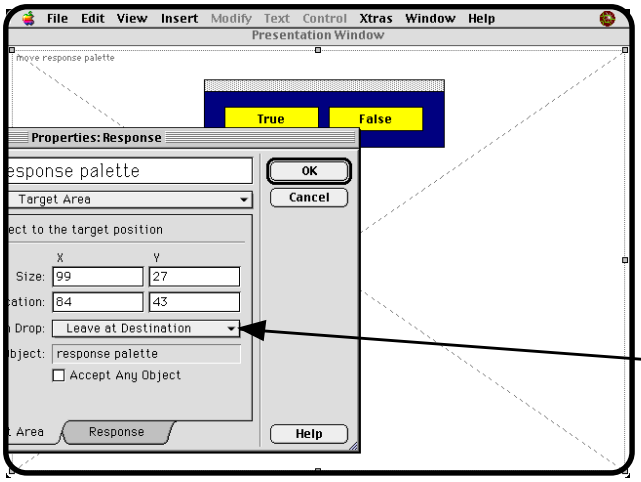
☐ **Click once on the response palette display.**

The target jumps directly to the middle of the response palette.



Notice also that the name of this display appears in the text box beside "Target Object" and a small image of the display appears in top left hand corner of the dialog box. We have just programmed Authorware to monitor if the response palette is moved into the target area. We can now move or resize the target area anyway we wish. Since we are simply interested if the user moves the response palette *anywhere* on the screen, we need to resize the target area so that it covers the entire screen. The target area acts like any other object. You can resize it by clicking and holding any of its four corners.

☐ **Resize the target area so that it covers the entire screen:**



Be absolutely sure that "Leave at Destination" shows here. If it doesn't, change it.

☐ **When finished, click "OK."**

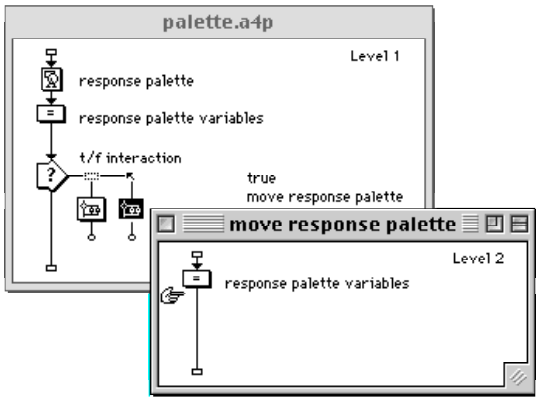
☐ **Press "Command-P" to proceed.**

All we have to do now is copy and paste the Calculation icon "response palette variables" into this target area branch and we are finished.

☐ **Jump to the flow line.**

☐ **Copy the Calculation icon "response palette variables."**

☐ **Paste the Calculation icon “response palette variables” to the Level 2 flow line inside the Map icon “move response palette”:**



That should do the trick. Let’s test it.

☐ **Restart the file and test it to be sure that no matter where you move the response palette, the hot spot is always behind the button display “True.”**

☐ **Save the file.**

Next steps

Construct a “close box” for the palette that erases the palette when checked.

Technique 5. Creating slide bars and other movable data objects

Introduction

The ambition of all software designers is to create a user interface that makes interaction on the computer as natural as possible. Buttons and hot spots are fine, but sometimes you just want to put your hand through the computer, grab an object and put it where you want it. While we can't design virtual reality with Authorware (at least not yet), we can take advantage of its capability to not only move objects on the screen, but also to have such manipulations change what a program does. Consider the following situation:

> You are designing a green house simulation. The simulation lets users change all kinds of conditions, such as the temperature, humidity, and also lets them add fertilizer, lime and water. You'd like the interface to be as realistic as possible. For example, you plan on having a picture of a thermostat and want users to be able to slide the setting up or down to adjust the temperature.

The point of designing the thermostat interaction in this way is that the user already knows immediately what to do when they see such a graphic — they all have used a similar thermostat in their homes. (In contrast, consider their reaction to cumbersome alternatives such as buttons that say "click here to increase the temperature 5 degrees.") Interactions designed with a more natural interface will let users focus on the goals of the simulation, not the mechanics of operating the program. As another example, recall that at the end of chapter two we discussed how great it would have been to let the user drag the offensive basketball player to different spots on the screen. The technique demonstrated in this section will let you do just that.

This section presents a very short example to demonstrate the concept. We will use the green house scenario above and build a working thermostat. The user will be able to move the thermostat's slider to whatever temperature they want and subsequently pass this information back to Authorware in the form of a variable (which could then be used by the simulation's "engine" to affect changes in plant growth).

Getting started

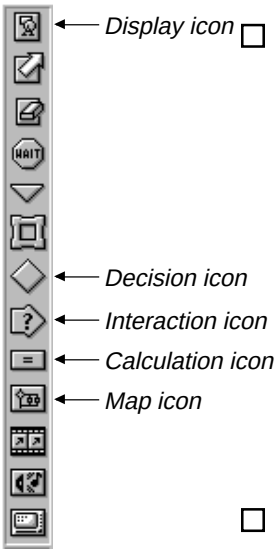
☐ **Launch Authorware and open a new file.**

☐ **Save the file with the name "thermo.a4p."**

Remember to save the file periodically through this session.

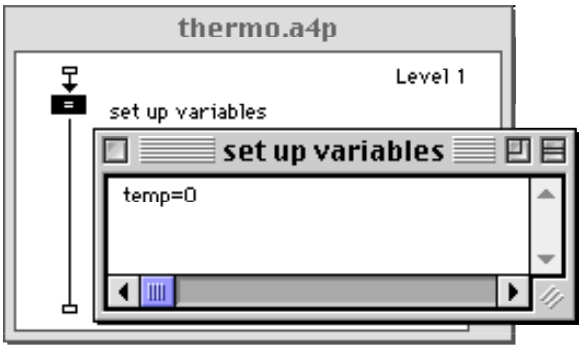
Let's start by setting up the a variable that will contain the current temperature indicated on the thermostat. As the user moves the thermostat's slider, this variable will change accordingly.

Here are the icons you will be using in this section:



☐ **Drag a Calculation icon to the flow line and title it "set up variables."**

☐ **Double-click on this Calculation icon and type "Temp=0" inside the text window:**

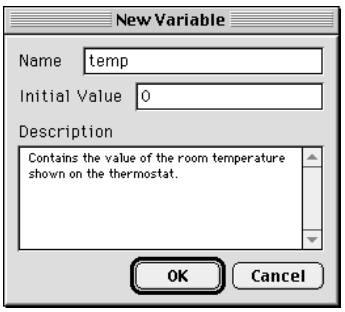


☐ **When finished, click in the close box in the Calculation icon's window.**

☐ **Click "Yes" to save the calculation changes.**

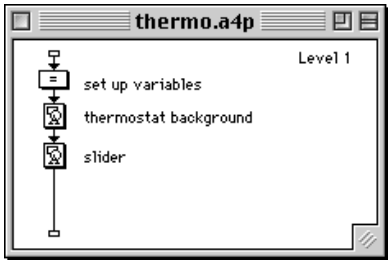
The New Variable dialog box will open.

☐ **Enter an initial value of "0" and a suitable description, then click "OK":**



Next let's construct the displays needed by this demonstration. Only two Display icons are needed — one for the background graphic of a thermostat and the other for the graphic of the slide bar. The slide bar must be contained within a separate Display icon because only it will be movable.

☐ **Drag two Display icons to the flow line; title the first "thermostat background" and the second "slider."**

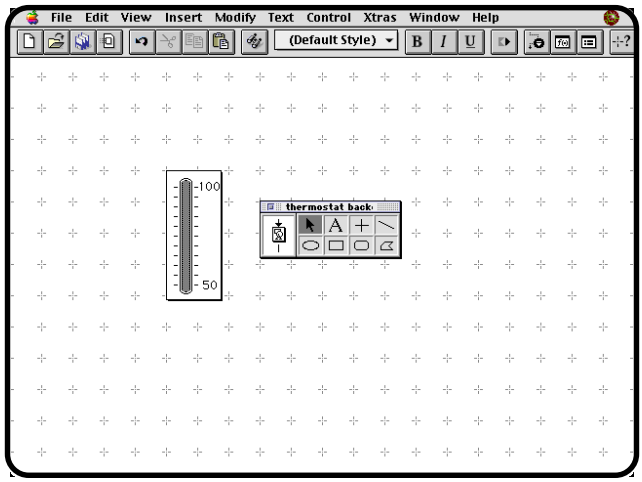


As has been our custom in this book, we will use simple graphics for both displays (these can be found in the Rieber Clip Art resource). We will use the trick of running the file to trigger the opening of these two icons. That way, the thermostat background will stay on the screen as we place the slider.

☐ **Restart the file.**

The Display icon "thermostat background" automatically opens.

☐ **Construct the following graphic (or copy and paste it from the Rieber Clip Art resource):**



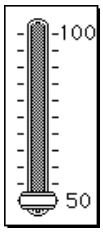
☐ **When finished, click in the close box in the tool box.**

The Display icon “slider” automatically opens.

☐ **Construct a simple graphic of the slide bar (or copy and paste it from the Rieber Clip Art resource):**

☐ **Align the slide bar carefully with the thermostat so that it is centered perfectly at the 50 degree mark:**

My apologies to non-American readers for my use of the Fahrenheit temperature scale. Using the scale with which most users are familiar is definitely an important user interface decision.

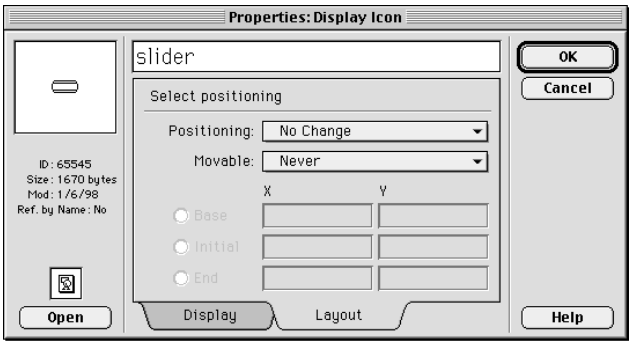


Next, we need to program Authorware to allow the user to move this slider and also to constrain its movement to the screen area indicated by the thermostat background.

☐ **Select “Icon” from the “Modify” menu, then select “Attributes...”.**

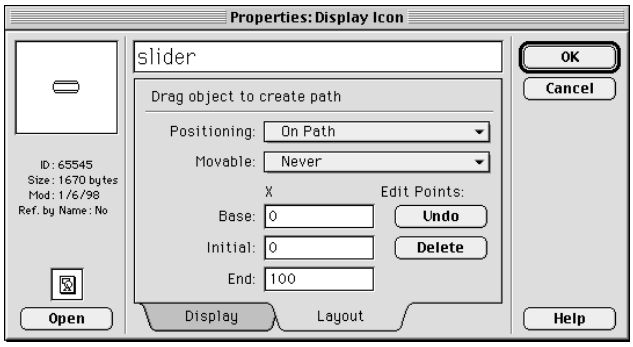
☐ **Click once on the Layout tab to view the Layout options.**

The following dialog box appears:



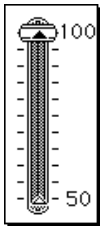
☐ **Change the Positioning option to “On path”.**

When you do, the dialog box changes giving you additional options:



As the directions in the dialog box indicate, you next need to drag the object to create the path. You may need to move the dialog out of the way for the next step.

☐ **Carefully move the slide so that it is perfectly centered at the 100 degree mark:**



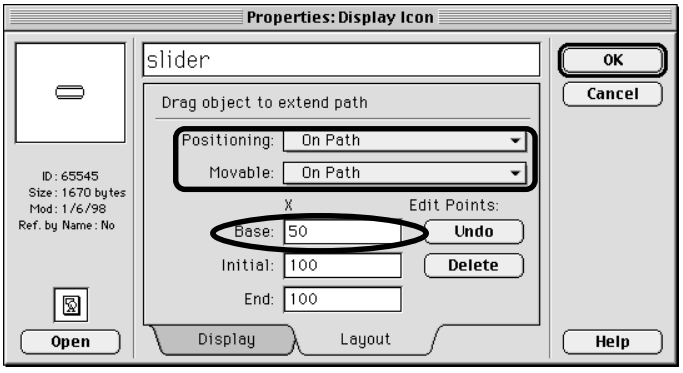
Next, we need to tell Authorware the values associated with the base and end positions of this path. The default is 0 for the base position and 100 for the end position. We need to change the base to 50.

☐ **Enter a value of "50" in the text box beside "Base".**

The last step is to program Authorware to let the user move this slide bar, but only along the path indicated.

☐ **Change the Movable option to "On path".**

Your dialog box should match the following:



We're done.

☐ **Click "OK."**

This sends us back to the open Display icon "slider."

☐ **Click in the close box in the tool box to close this icon.**

Recall that we were running the file at the time, so the program resumes execution. Since there are no more icons on the flow line, the program hangs. We can now test to see if the slider is movable only along the path defined in the previous steps. However, it is important to restart the file first.

☐ **Restart the file by running it again** (by pressing "Command/Control-R")**.**

Let's test it.

☐ **Move the slide bar up and down anywhere on the thermostat.**

However, you should not be able to move it anywhere else.

Manipulating data with the slide bar

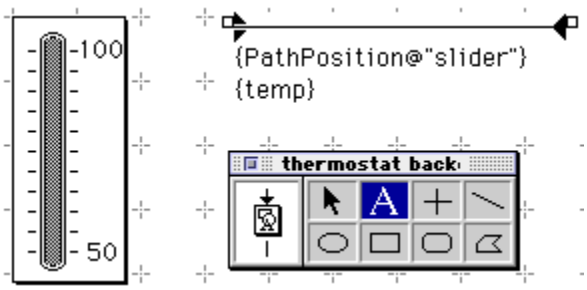
As you know, the goal is to be able to use the slide bar interactively within a simulation, such as the greenhouse context. As we adjust the temperature with this thermostat, we will be altering the climate conditions inside the greenhouse. Therefore, the next step is to update the variable "temp" to match the position of the slide bar each time it is moved. This is done with the system variable "PathPosition@IconTitle." Let's see how this works by adding it to our program.

☐ **Restart the file.**

☐ **Pause the file** (by pressing Command/Control-P)**.**

☐ **Double-click on the thermostat background graphic to open its Display icon.**

☐ **With the text tool, type the following:**



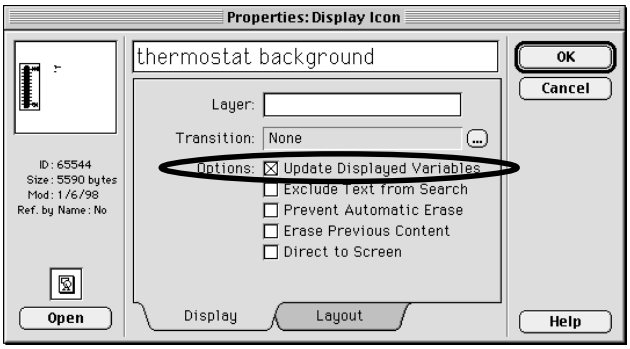
PathPosition@"IconTitle" contains the position of the display indicated by “IconTitle” (the Display icon “slider”) on the path that we have already programmed. By showing the contents of these variables here, we can test to our program to make sure it is functioning up to this point. As you will soon see, the variable “temp” is still not functioning properly (but we will correct it soon).

Of course, we need to tell Authorware to update all the variables displayed in the “thermostat background.”

☐ **Select “Icon” from the “Modify” menu, then select “Properties...”**

☐ **Click once on the Display tab to view the Display options.**

☐ **Turn on “Update Displayed Variables”:**



☐ **Click “OK.”**

☐ **Restart the file by running it again (by pressing “Command/Control-R”).**

☐ **Test the program by moving the slide bar to different locations, noting whether the first number to the right changes appropriately.**

Again, don’t be alarmed if the second number stays zero. We have yet to program that part. Focus attention on how the first number changes as you move the slider. If the num-

bers seem a little off, it's probably because you did not perfectly align the slider with the 50 and 100 tick marks. You can adjust the slider by opening its Display icon, selecting "Icon" from the "Modify" menu, then selecting "Properties...", then adjusting the slider's path.

☐ **Jump back to the flow line.**

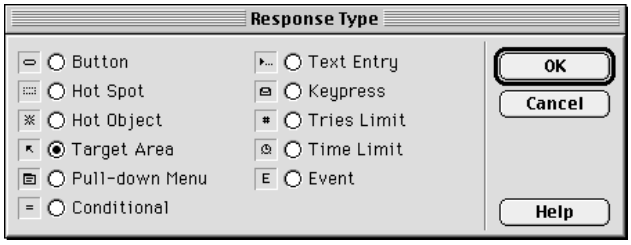
Transferring this data to the variable "temp"

There is one important final step. We need to "give" the data currently held by PathPosition@"slider" to the variable "temp." Once we have it stored in "temp" we can use it in to alter our simulation. We will accomplish this by setting up a perpetual interaction using the "target area" response type. As discussed in the previous section where we constructed a movable response palette, this response type allows us to tell Authorware to perform an action when it notices if a particular screen object is moved.

☐ **Drag an Interaction icon to the bottom of the flow line and title it "thermostat interaction."**

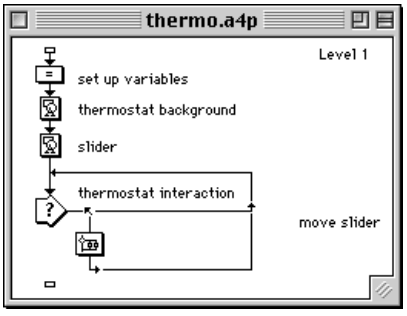
☐ **Attach a Map icon to this Interaction icon.**

☐ **When the Response Type dialog box automatically opens, choose "Target Area," then click "OK."**



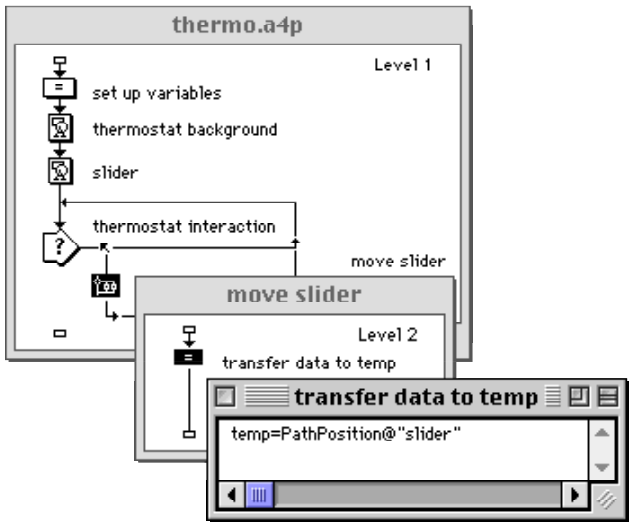
☐ **Title the branch associated with this Map icon "move slider."**

Your flow line should look like this:



Next, we will embed a Calculation icon in this Map icon in which we will transfer the data from PathPosition@"slider" to the variable "temp."

- ☐ **Double-click to open the Map icon "move slider."**
- ☐ **Drag a Calculation icon to this Level 2 flow line and title it "transfer data to temp".**
- ☐ **Open this Calculation icon and type this line: temp=PathPosition@"slider"**



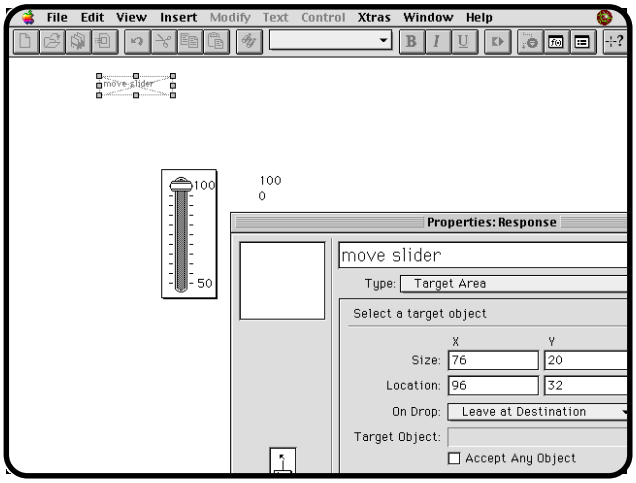
In human terms, this line tells Authorware to make the variable "temp" equal to the current value of the slider within its path on the thermostat.

- ☐ **When done, click in the close box in the Calculation icon's window.**
- ☐ **Click "Yes" to save the calculation changes.**

Now we need to program Authorware to detect if the user has moved the response palette within the thermostat. We can do this just by running the file.

- ☐ **Restart the file.**

Authorware pauses the program automatically because it notices that the "Target Area" action is yet undefined:

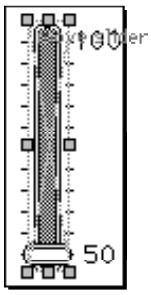


Let's follow the directions in the dialog box: "Select a target object".

☐ **Click once on the slider graphic.**

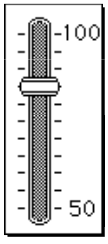
The target area jumps directly to the middle of the slider graphic and the word "slider" now appears at the end of the directions in the black bar indicating that Authorware has been programmed to associate this display with this target area.

☐ **Resize the target area to match the slider's path on the thermostat:**



☐ **Be sure that "Leave at Destination" shows in the dialog box.**

☐ **Click "OK."**



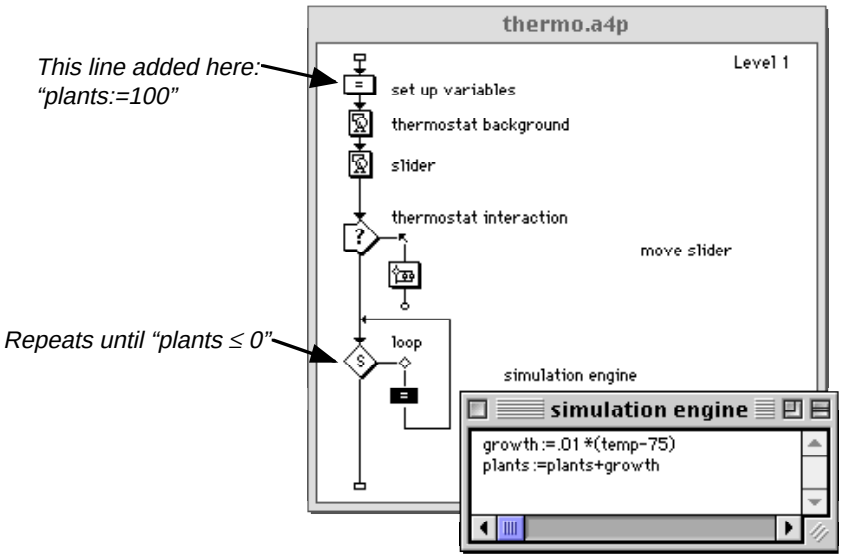
85
85

☐ **Test the program by moving the slider.**

This time, the variable "temp," previously embedded on the thermostat background, should also update itself on the screen as soon as you release the mouse button after dragging it to a new position.

Using the data to alter conditions in a simulation

Let's very briefly consider what you might do given the ability to control a program's data via a movable slider. In our greenhouse simulation, the idea is that temperature has some effect on plant growth. Let's say we are trying to grow a tropical plant species that thrives in warmer temperatures. We would like the simulation to demonstrate this relationship for the user — the plants will grow when the temperature is warm and die if the temperature is too cold. Of course, heating the greenhouse (especially if it's in a cold climate) is expensive, so the goal for users would be to try to find an optimal situation in which we get the most plant growth from the least amount of heat. Obviously, this is a complex relationship to program. However, let's take a look at the first step: building a simple linear relationship between plant growth and temperature. For example, consider the following modifications to our program:



As you can see, I changed the Target Area response branch to a perpetual interaction and also added a loop (just like that used in several previous chapters). The Calculation icon "simulation engine" contains the following two lines:

```
growth=.01*(temp-75)
plants=plants+growth
```

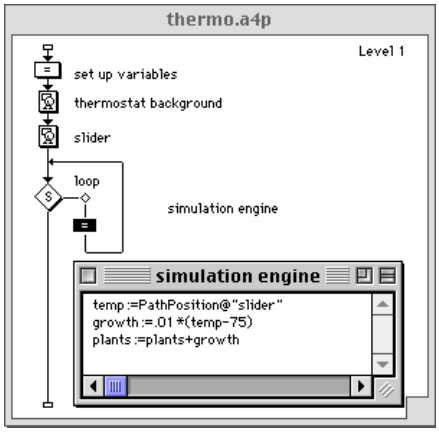
The first line is something I just conjured up using my own imagination (my apologies to real botanists and biologists). It sets a variable called "growth" to a positive number when the temperature is above 75 degrees (i.e. plants grow) and a negative when it is below 75 degrees (i.e. plants die). No growth takes place when the temperature exactly equals 75 degrees. I multiply everything by .01 just to slow down this

rate of change. The second line uses the variable “growth” to adjust the total number of plants we have in the greenhouse (via a new variable named “plants”).

Assuming we started out with 100 plants, these two lines would allow users to manipulate the temperature setting and watch the outcome. The program could simply display the total number of plants currently in the greenhouse, updating it after each loop (similar to how we currently display the temperature inside the Display icon “thermostat background”). Another option would be to have the variable “plants” control a series of graphics that illustrate the increase or decrease in plants in visual form (such as a QuickTime movie).

Another strategy

You might wonder why it is even necessary to use an Interaction Icon in tandem with a Target Area response type in the first place. The PathPosition@”IconTitle” obviously updates itself continuously. It seems as though there must be another way to transfer the data to the simulation. Yes, it is possible to construct an interaction without resorting to the use of an Interaction icon in conjunction with the Target Area response type, but it would require another approach for transferring the data associated with the PathPosition system variable to be used by the simulation. For example, the loop associated with the “simulation engine” that we just discussed would work if we moved the programming line “temp:=PathPosition@”slider” from the Target Area response and added inside the “simulation engine.” The data would be passed from PathPosition@”slider” to “temp” at every loop:



In a sense, both approaches act as “triggers” for telling Authorware when to take the data from the PathPosition system variable and use it in the simulation. I have found the approach using the Target Area response type to be a good general strategy that works well in most designs since it is not dependent on a loop.

Technique 6. Collecting and analyzing data with a spreadsheet

Introduction

Most of the projects we have built in this book have kept track of numerous bits of information, such as game scores. They all have the potential for collecting lots of other interesting information as well, such as time spent playing the game and number of questions answered correctly and incorrectly. However, as soon as the player quits the computer game, all of these data are lost. Frankly, all educational applications demand excellent record keeping since teachers are held accountable for the way in which the school day is spent. Computer software that does not have record keeping features may create an unreasonable amount of extra work for teachers as they try to monitor what software was used and what was accomplished. However, there are also lots of student-centered reasons for knowing how to collect and analyze data by computer. Consider the following situation:

> You're helping Lisa, a high school senior, with a project for the local math and science fair. She has recently come across some research literature in psychology, anthropology, and sociology that discusses several theories explaining gender differences in game playing behavior. Based on these theories, most of the popular games seem predominantly aimed toward males. Lisa is also a volunteer at the local library. Over the last year, the library has invested in six Macintosh computers and has bought a small collection of computer software — all in the "edutainment" category. The computers and software seem very popular with the library patrons. The computers are always in use when Lisa arrives right after school and seem to stay busy until she leaves to go home at about 7:00 p.m. However, it always seems as though only young males are the ones using them. Lisa wonders whether her informal observations are accurate. She also wonders whether the theories she has been reading about are accurate reflections of game playing behavior in the local community. The decision to buy the next allotment of software with this year's budget will soon be made by the library staff. Lisa has persuaded them to conduct a survey of game playing behaviors and preferences of library patrons over the next two weeks, the results of which they hope will help inform their decision. The survey will also be the cornerstone of her math and science fair project. Since she can only be at the library for a

few hours each week, she has decided to computerize the survey so that it is self-running and automatically saves the data to disk. She convinced the library staff to dedicate one of the computers for the survey for the next two weeks.

Lisa's project lends itself to all sorts of interesting questions — some she will ask on her survey and some she needs to ask herself as she designs her research project. For example, she will need to consider whether or not all library patrons should be allowed to answer the questions or only those using the library's computers. She needs to consider whether simply allowing patrons to voluntarily complete the survey is an adequate sampling technique. Perhaps only those who like games will complete it, thus skewing her data inappropriately. She also needs to make sure that someone doesn't complete the survey more than once. Similarly, how is she going to prevent a prankster from repeatedly entering bogus data? We'll keep our fingers crossed and hope that Lisa can work out all the procedures for collecting a representative sample for her project. We want to turn out attention to the mechanics of setting up a survey such as Lisa's.

Besides the mechanics of setting up the questions and variables (procedures with which you should feel comfortable with by now), there are really just two major components to a survey such as Lisa's. The first is the procedure for having Authorware collect the data and write it to a text file on a disk. The second is reading and analyzing the data. Frankly, it is the second component that demands most of the programming skill. Writing code that will do even the simplest of analyses, such as sorting the data by gender, then compiling basic descriptive statistics, such as number of entries, averages across questions, etc. is exceedingly demanding and tedious requiring mastery of some very sophisticated programming techniques, such as arrays and strings. Fortunately, we are going to skip this component entirely without sacrificing any of the data's results. Instead of using Authorware (or some other authoring tool) for analysis, we are going to save the data in a form that will lend itself to copying and pasting into any commercial spreadsheet application, within which we can perform as complex a data analysis as we wish. The trick is to save the data into the "row and column" format in which all spreadsheets operate.

Even the most powerful data analysis programs, such as SPSS and SAS, operate on the row and column format.

In the "row and column" format, each row contains one of the survey respondents, and each column contains one of the data sources we are collecting (such as the answer to each question on the survey). Therefore, each line of data corresponds to one respondent. Each piece of data in the line corresponds to one single variable separated by a Tab key. A Return key is inserted at the end of the data line so that the next respondent's data will begin on the next line. Authorware will write this data to a disk in the form of a simple text file adding one line at a time. As soon as one

respondent finishes the survey, one line of data will be added to the data file. In this way, the data file will grow slowly over time. At any point, we can open the text file and copy and paste all of the data into the spreadsheet, including the embedded Tab and Return keys. The data will then neatly assign itself automatically to one variable per cell and one respondent per row. Finally, we can use the spreadsheet's many functions to analyze the data anyway we wish. Obviously, this technique assumes you are well versed in the use of a spreadsheet.

More about the data structure

To get a better understanding of the eventual data structure that we are aiming for, consider a simple survey that asks three questions:

1) Are you male or female?
2) How old are you?
3) How many hours each week do you play computer games?

It would also be nice to know the date and time that the person answered the survey. Fortunately, we can have Authorware add these two pieces of data for us by having it check the computer's system clock (more about these in just a moment). So, if six people answered the survey over the span of three days, the text file might contain data similar to the following:

	Tab	*Tab*	*Tab*	*Tab*	*Return*
male	11	10	11/22/96	3:32 PM	
male	15	25	11/22/96	3:41 PM	
female	13	6	11/22/96	3:58 PM	
male	24	3	11/23/96	6:48 PM	
male	18	7	11/23/96	8:12 PM	
female	9	4	11/24/96	9:09 AM	

Tab keys are inserted between each piece of data and a Return key is inserted at the end of each line.

As you can see, each line refers to one person and shows their three answers to the survey followed by the date and time they completed it. If these data were copied and pasted into a spreadsheet, we could then perform any number of analyses, such as counting up the number of males and females and computing their average age and the average amount of time they report playing computer games. We will also see some patterns associated with when people were filling out the survey. Of course, there are many other important and interesting questions we could ask. However, these will suffice for the purposes of this demonstration.

In our Authorware program, we will create three variables for each of the three questions: "gender," "age," and "usage." We can get the date and time from two system variables named (not surprisingly) "Date" and "Time." When the respondent finishes the survey, we need to "assemble" these individual variables, along with Tabs in-between and a Re-

turn at the end, into one long string through a process called concatenation.

For example, consider the following line of code embedded in a Calculation icon:

```
results :=gender^Tab^age^Tab^usage^Tab^Date^Tab^Time^Return
```

A good analogy for concatenation is addition. Think of the ^ symbol as being like the + symbol, but instead of adding numbers, you are adding words.

The symbol ^ is known as the concatenation operator. With it you can join two or more strings together into one string. As you can see we are joining the five variables together with a Tab in-between each and a Return at the end. This one long string is then stored in the variable "results" which, in turn, is saved to a text file on the computer via the AppendExtFile("filename", string) function. As its name implies, this function appends, or adds to, the text file on the disk. For example, we will embed the following in a Calculation icon at the very end of the program:

As compared to WriteExtFile("filename", string) which overwrites the contents of the specified text file with the contents of the string.

```
AppendExtFile(FileLocation^"survey results",results)
```

The name of the text file being saved to disk is "survey results" and to it Authorware appends the contents of the variable "results" (containing all of the data for the person who just completed the survey). An important question is "Where on the disk is Authorware saving the text file?" Although it is possible to save the text file "survey results" anywhere on the disk, for simplicity we will choose to just save it to the same disk location that the survey program itself is located. Fortunately, Authorware has another system variable named "FileLocation" that holds this very information. As you can see, we need to concatenate this system variable to the name of the text file.

Hopefully, this gives you a good orientation to what needs to be done. You'll understand it better by doing it.

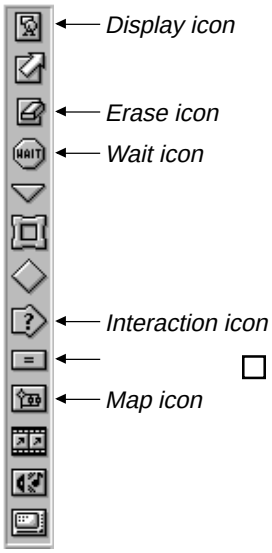
Getting started

- ☐ **Launch Authorware and open a new file.**
- ☐ **Save the file with the name "survey.a4p".**

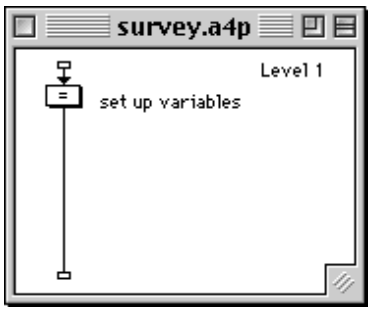
Remember to save the file periodically through this session.

Let's begin by setting up all the variables we are going to use in this demonstration. We will keep our survey prototype simple by only asking the three questions about the respondent's gender, age, and amount of game playing per week (in hours). Obviously, Lisa would want to ask a lot more questions for her project.

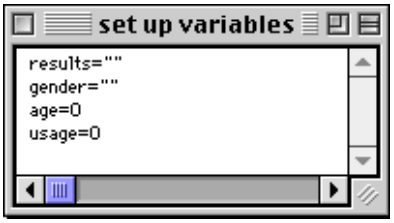
Here are the icons you will be using in this section:



☐ **Drag a Calculation icon to the flow line and title it "set up variables:"**



☐ **Double-click on this Calculation icon and enter the following four lines:**



The variables "results" and "gender" are character variables — they will contain nonnumeric information. For example, "gender" will contain either the character string "male" or "female." The variables "age" and "usage" are numeric variables, the first will keep track of the respondent's age and the second will keep track of the number of hours the respondent indicates are spent playing games each week. The variables "results" will be used to store the contents of the other three variables via the concatenation process. The result of this process is a character variable (more about this process later). One assigns the content to a character variable with quotes around the contents. In the above example, we are setting "gender" and "results" to an empty string.

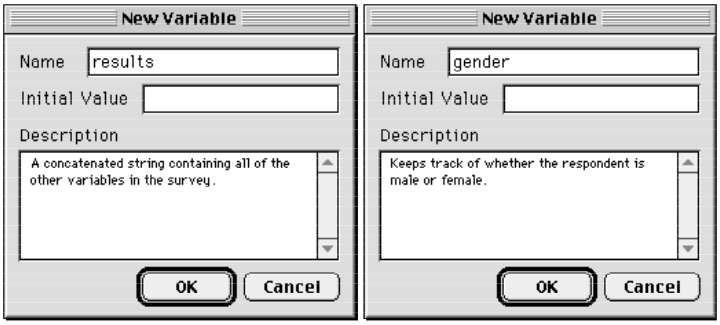
Similar to the idea of assigning 0 to a numeric variable just to set up the program's variable structure.

☐ **When finished, click in the close box in the Calculation icon's window.**

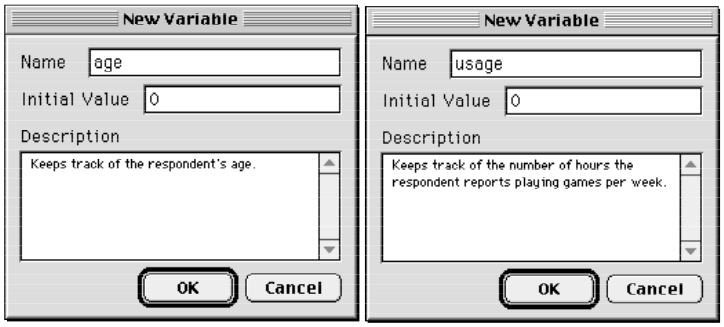
☐ **Click "Yes" to save the calculation changes.**

The New Variable dialog box will open for each variable.

☐ **For the character variables “results” and “gender,” leave the initial value box blank and enter a suitable description (such as the following), then click “OK”:**



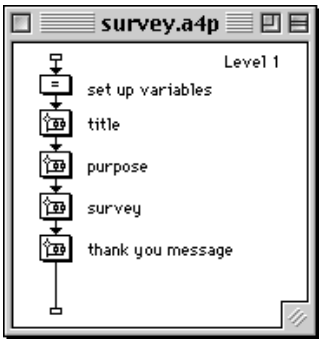
☐ **For the numeric variables “age” and “usage,” enter an initial value of 0 and a suitable description (such as the following), then click “OK”:**



Constructing the survey’s file structure

Next, let’s construct the general structure to this survey.

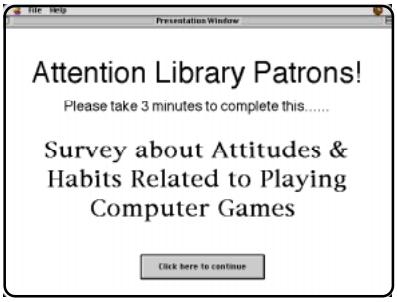
☐ **Drag four Map icons to the flow line and title them “title,” “purpose,” “survey,” and “thank you message” respectively:**

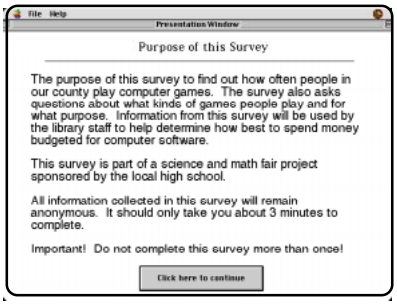


☐ **Construct suitable title and purpose frames for the survey in each of the respective Map icons.**

It's assumed that you know how to do this. Here are some very simple title and purpose frames you might use as samples for now:

The design of a survey is not a trivial matter. When well constructed, it should take the respondent no longer than necessary to complete. In fact, since most surveys conducted as part of research projects are completed on a voluntary basis, many potential respondents will probably choose not to participate if the survey looks too confusing or grueling.





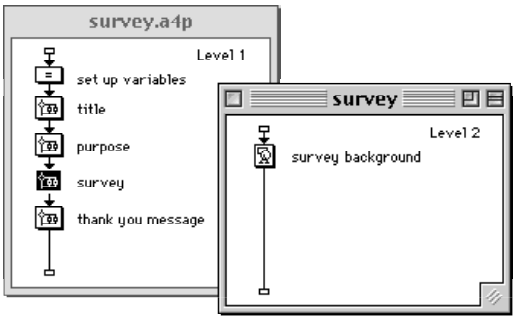
Let's turn out attention to the constructing the survey questions.

Constructing the survey questions

As you know, this survey will ask three questions. Each will be constructed with an Interaction icon. Let's begin by setting up a background display that will be shared by all three questions.

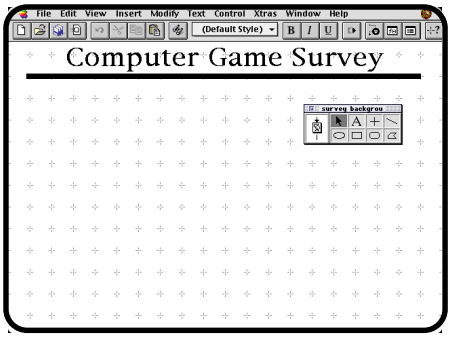
☐ **Double-click to open the Map icon "survey."**

☐ **Drag a Display icon to the Level 2 flow line and title it "survey background":**



☐ **In this Display icon, construct a background that all three questions will share.**

For example, I constructed a very simple background consisting of just a general title at the top:



Setting up a multiple-choice question for gender

Since the answer to this question is either "male" or "female," it makes sense to set up two pushbuttons.

☐ **Drag an Interaction to the flow line and title it "gender question."**

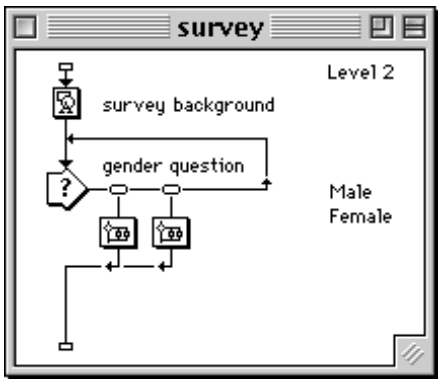
☐ **Construct two branches off of this Interaction icon with Map icons; use the pushbutton response type for each.**

☐ **Title the first branch "Male" and the other "Female."**

☐ **Change each branch to "exit interaction."**

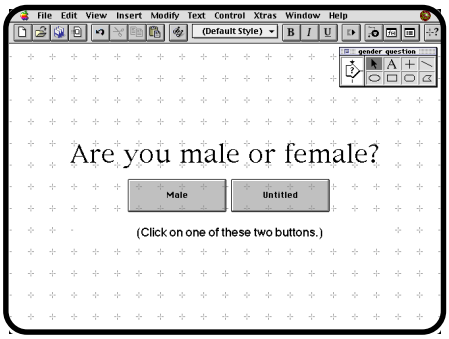
Reminder! *The shortcut to doing this is "Command/Control-clicking" on the flow line coming out of each Map icon.*

When done, your interaction should look like this:



Now let's set up the display for this question.

☐ **Double-click on the Interaction icon "gender question" and construct the following display:**

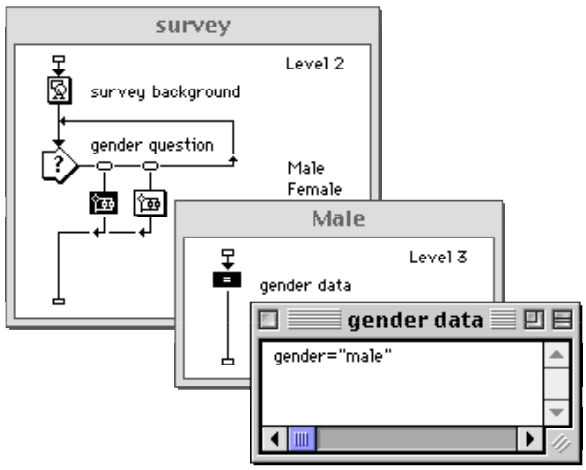


☐ **Close this Interaction icon when done.**

Next, we need to embed a Calculation icon inside each of these two Map icons that assign either "male" or "female" to the variable "gender."

☐ **Open the Map icon "Male," add a Calculation icon to the Level 3 flow line and title it "gender data."**

☐ **Open this Calculation icon and type the following line: gender="male"**

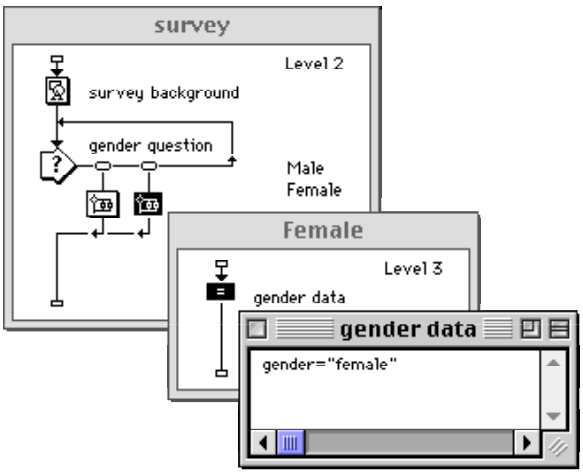


☐ **Close the Calculation icon.**

Now we repeat the process for the Map icon "Female." However, we will assign "female" to the variable "gender" instead.

☐ **Open the Map icon "Female" and add a Calculation icon to the Level 3 flow line, also titling it "gender data."**

☐ **Open this Calculation icon and type the following line: gender="female".**



☐ **Close the Calculation icon.**

Let's move on to the next question which asks for the respondent's age.

Setting up a text entry question for age

☐ **Drag an Interaction to the flow line and title it "age question."**

The best response type for this question is text entry. This will allow the respondent to type in their age from the keyboard.

☐ **Construct one branch off of this Interaction icon with a Map icon; select the text entry response type for this branch.**

As first described in Chapter 3, the title for a text entry response branch is very special. It acts as a match against which Authorware compares the user's response. In this way, you can set up open-ended questions, such as "Who was the first person to walk on the moon?" Users who type "Neil Armstrong" vs. "Richard Nixon" would be given different feedback. However, in our case, we want to accept any answer that the user types in, then extract the first number that we find. For that reason, we are going to title this branch with a special wildcard symbol — the asterisk (*) — that tells Authorware to match on anything.

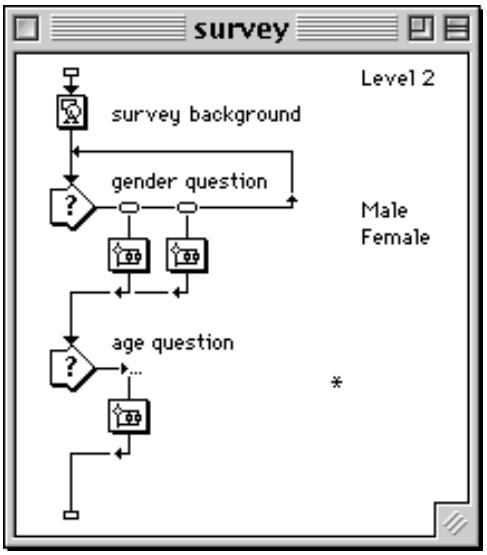
You are encouraged to read the Authorware reference manuals for a complete description of how the text entry response type works.

☐ **Title this text entry response branch "*".**

☐ **Change this branch to "exit interaction."**

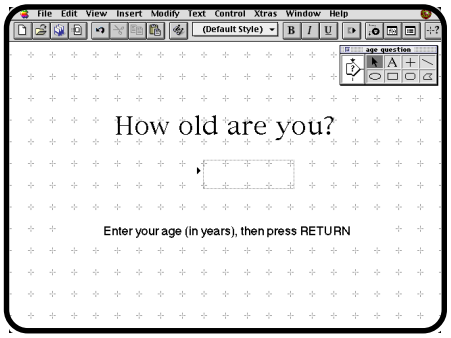
Also make sure that it is set to "Erase Interaction: Upon Exit." (This is the default choice for this option.)

When done, your interaction should look like this:



Now let's set up the display for this question.

☐ **Double-click on the Interaction icon "age question" and construct the following display:**



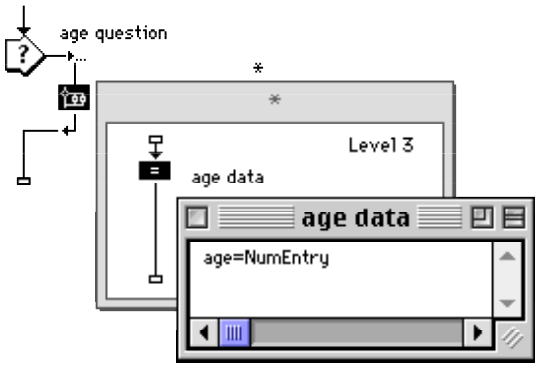
Notice the fuzzy box in which the user's answer will be typed. This will not be visible when the program is run. You can move and resize it.

To change the font, font size, style, or color, just double-click on the text box to open the text field dialog box. These changes can be made in the Text options.

☐ **Close this Interaction icon when done.**

☐ **Open the Map icon "*," add a Calculation icon to the Level 3 flow line and title it "age data."**

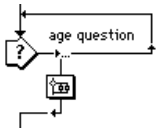
☐ **Open this Calculation icon and type the following line: age=NumEntry**



NumEntry is a system variable that contains the first number the user types in response to the most recent text entry interaction. This line tells Authorware to assign this value to the variable "age."

☐ **Close the Calculation icon.**

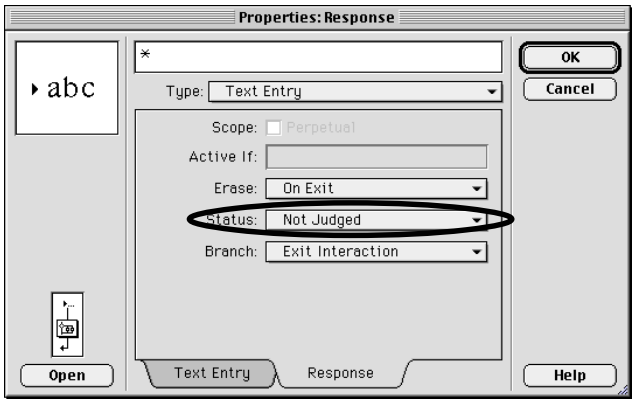
We want to erase the respondent's answer to this question as soon as the Return key is pressed. This can be done by choosing "Erase Feedback: On Exit" within the Text Entry Options dialog box.



☐ **Open the Text Entry Options dialog box by double-clicking on the text entry symbol ( ) leading into the branch.**

☐ **Click on the Response tab to view the Response options.**

☐ **Change the "Erase" option to "On Exit."**



☐ **Click "OK".**

Setting up a text entry question for game playing

We set up the third and final question in a way almost identical to the second question since we again are asking the user to type in a number.

☐ **Drag an Interaction to the flow line and title it ”game playing question.”**

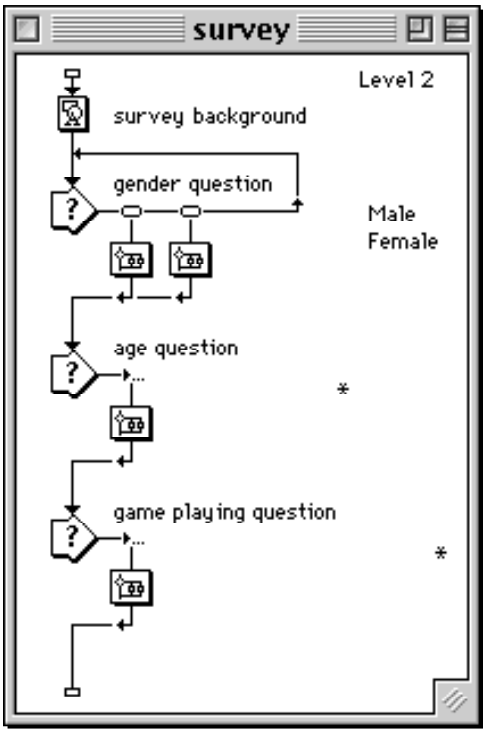
We will again use the text entry response type for this question.

☐ **Construct one branch off of this Interaction icon with a Map icon; select the text entry response type for this branch.**

☐ **Title this text entry response branch “*”.**

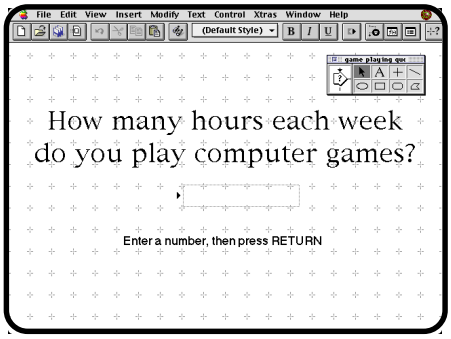
☐ **Change this branch to “exit interaction.”**

When done, your interaction should look like this:



Now let's set up the display for this question.

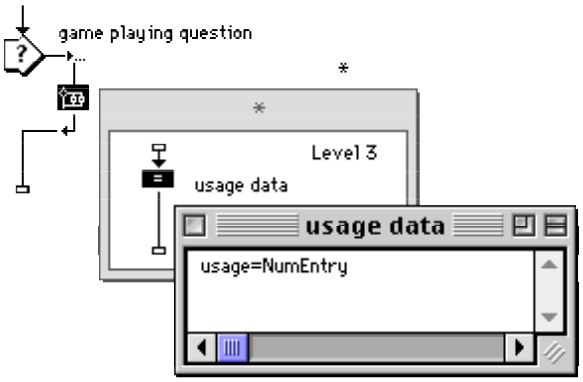
☐ **Double-click on the Interaction icon "game playing question" and construct the following display:**



☐ **Close this Interaction icon when done.**

☐ **Open the Map icon "*," add a Calculation icon to the Level 3 flow line and title it "usage data."**

☐ **Inside this Calculation icon, add the following one line: usage=NumEntry**



☐ **Close the Calculation icon.**

We again want to erase the respondent's answer to this question as soon as the Return key is pressed.

☐ **Open the Text Entry Options dialog box by double-clicking on the text entry symbol ( ) leading into the branch.**

☐ **Click on the Response tab to view the Response options.**

☐ **Change the "Erase" option to "On Exit."**

☐ **Click "OK".**

The survey background display will need to be erased at the end of the survey.

☐ **Drag an Erase icon to the bottom of the Level 2 flow line inside the Map icon "survey" and title it "erase survey background."**

We'll program this Erase icon to erase the survey background in a few minutes when we run the file. For now, let's just move on and finish the job of setting up all the variables. There is just one more variable to go — the variable "results."

Concatenating the survey data

When the respondent has finished answering the survey questions, we need to have Authorware concatenate, or join together, all of the survey's variables into one long string and assign it to the variable "results."

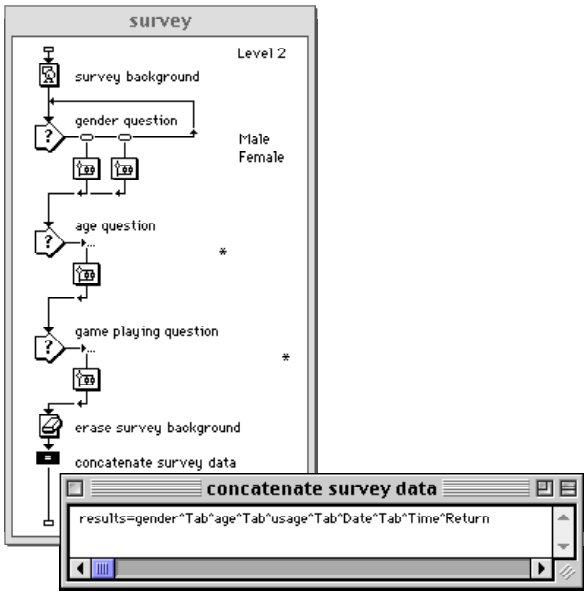
☐ **Drag a Calculation icon to the bottom of the Level 2 flow line inside the Map icon "survey" and title it "concatenate survey data."**

☐ **Open this Calculation icon and type the following line:**

```
results=gender^Tab^age^Tab^usage^Tab^Date^Tab^Time^Return
```

The concatenation operator (^) is typed by pressing Shift-6.

It's important to note that the variable "results" is a character variable, even though several of the variables being concatenated together are not (such as "age" and "usage"). The concatenation process transforms all the data into a string of characters.



"Date" and "Time" are system variables. For more information about them, select "Show Variables..." under "Data"; they are listed in the category "Time."

This programming line assigns the respondent's answers to the three survey question plus the current date and time to the variable "results." A Tab key is inserted in-between each of these five data elements to separate them and to make

the data compatible with a spreadsheet. The Return key is placed at the very end so that the next respondent's data will start on a fresh line when we save the data to a text file.

☐ **Close the Calculation icon.**

☐ **Save the file.**

Testing the program

Let's run the program to make sure all of the variables are working properly. This will also be a good opportunity to fine tune any of the displays. We will also take care of defining the Erase icon "erase survey background."

As we run the program we can check to see if the variables are working by pausing the program, then selecting "Variables" under the "Window" menu. Our four user variables will be listed in the category "survey.a4p," the same name as the file. From there we can click on any of the variables to check their current values.

☐ **Restart the file.**

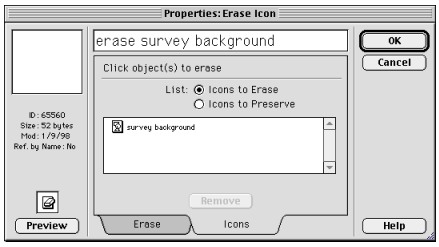
☐ **Before you answer the first survey question, pause the file (Command/Control-P) and check the current values of the four user variables, then proceed (press Command/Control-P again).**

The variables "age" and "usage" should have a current value of 0 and the variables "gender" and "results" should have a current value of an empty string (shown by "").

☐ **Go through the entire survey, answer the three survey questions as you go.**

When you get to the end, the Erase icon "erase survey background" will pop open.

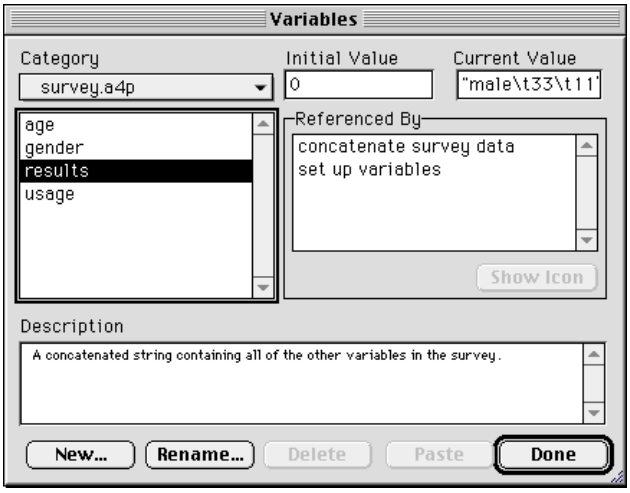
☐ **Click once on the survey background to erase it, then click "OK."**



This is the end of the file, so the program will hang when the screen goes blank.

☐ **Again check the current values of the four user variables.**

Each of the variables should show a current value equal to whatever you entered when you answered the survey questions. Be sure to check the variable "results." Its current value should contain the contents of each of the other variables separated by a forward slash (\). You should see a "t" for each of the Tab keys and a "r" at the end for the Return key:



Of course, this is a long string and the "Current Value:" window is not very big. Just click once inside this text window and use the right and left arrow keys on the keyboard to scroll through it.

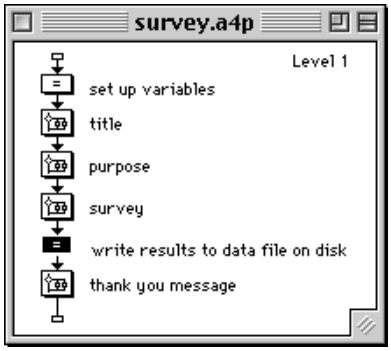
☐ **Jump back to the flow line.**

☐ **Save the file.**

Saving the data to disk

The next step is to save the data to the disk, appending the current respondent's answers to a text file.

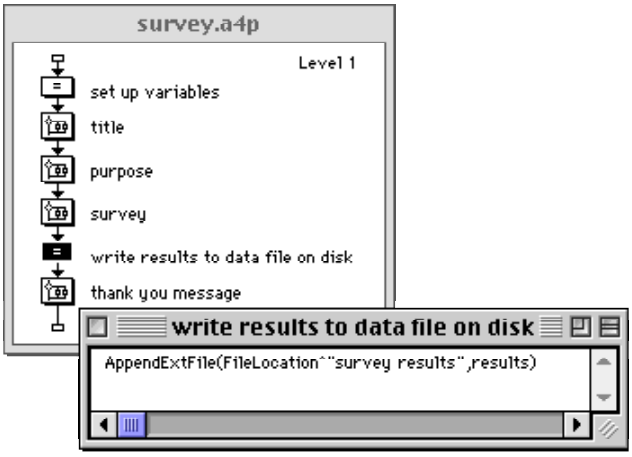
☐ **Drag a Calculation icon to the Level 1 flow line and put it between the Map icons "survey' and "thank you message"; title it "write results to data file on disk":**



☐ **Open this Calculation icon and add the following line:**

```
AppendExtFile(FileLocation^"survey results",results)
```

This function is in the File category of functions.



As previously discussed, the AppendExtFile("filename", string) function is used to add data to a text file (this function will also create the file if it doesn't already exist). We are telling it to append the string associated with the current value of the variable "results" to the text file named "survey results." By concatenating the system variable "FileLocation" to the beginning of the file name, we are telling Authorware to save this text file to the same folder (or directory) in which the Authorware file itself resides.

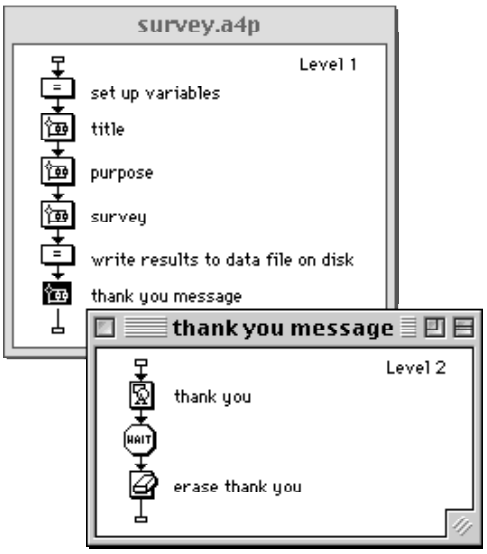
☐ **Close the Calculation icon.**

Thanking the respondent and restarting the file

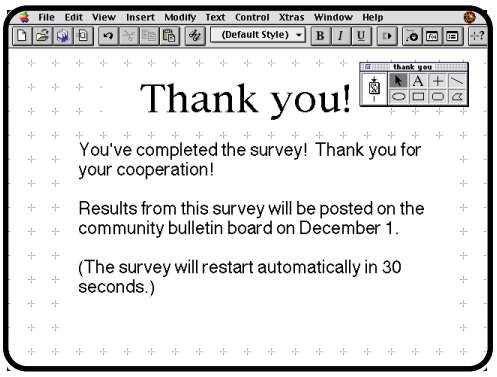
Before we test the program by running it and answering the survey several times to see if Authorware is really saving the data to the disk, let's quickly finish the file by adding one display that thanks the respondent and then restart the file automatically for the next user.

☐ **Double-click to open the Map icon "thank you message."**

☐ **Add a Display icon, Wait icon, and Erase icon to the Level 2 flow line, titling them as per the following:**



☐ **Open the Display icon "thank you" and construct a display similar to the following:**



☐ **Close this Display icon when done.**

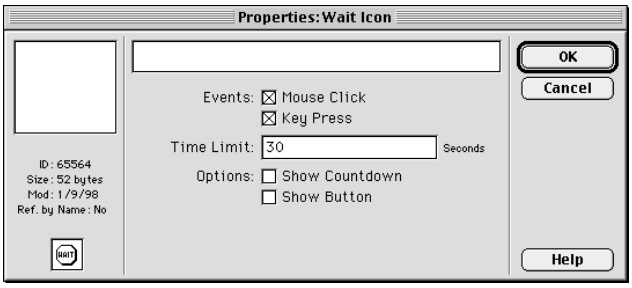
We might as well go ahead and program the Erase icon to erase this display.

☐ **Double-click to open the Erase icon "erase thank you"; click once on the display to erase it, then click "OK."**

As this thank you message indicates, we want the file to automatically restart after 30 seconds. That way, it won't be dependent on the respondent remembering to press any keys or click the mouse. This is easily accomplished by programming the Wait icon.

☐ **Double-click on the Wait icon to open the Wait Options dialog box.**

☐ **Enter a time limit of 30 seconds; select the "Mouse Click" and "Key Press" events; turn off the "Show Button" option:**



☐ **Click "OK."**

All that's left to do is embed the Restart function into a Calculation icon.

☐ **Add a Calculation icon to the very bottom of the Level 1 flow line and title it "restart."**

☐ **Open this Calculation icon and add the following line: "Restart()"**

☐ **Close the Calculation icon.**

☐ **Save the file.**

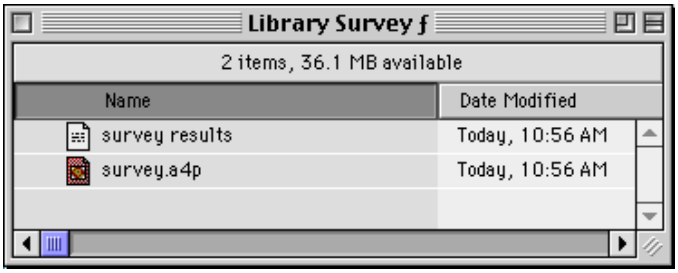
Checking the data file, then copying and pasting it into a spreadsheet

Now for the acid test! We need to make sure that the file is saving each respondent's answers to the question to the disk, appending them to the rest of the text file. Let's run the file and answer the survey questions several times. Then, we will check to see if the data file is on the disk and check its contents.

☐ **Restart the file.**

☐ **Go through the entire survey at least three times. Be sure to answer the survey questions a little differently each time.**

You should now have a text file saved to your computer's hard drive in the same folder or directory as your Authorware file:



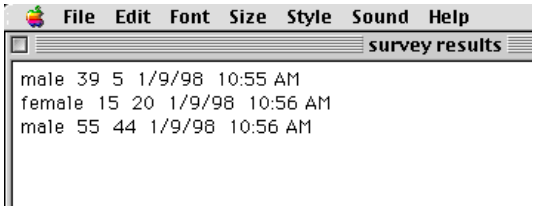
Windows Alert!

This is the Macintosh desktop, so obviously your screen will look very different.

Let's open it and take a look inside.

☐ **Go to your computer's desktop and open the text document "survey results."**

You should see the survey question data entered along with the date and time:



This is being viewed on a Macintosh with a text editor called SimpleText (included free on every Macintosh). You will need to open the text file "survey results" with a similar text editor or word processor.

Now let's copy and paste this data into a spreadsheet.

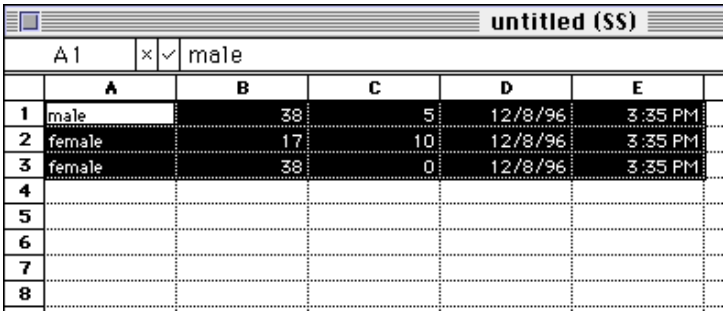
☐ **Highlight all of the data and copy it to the clipboard.**

☐ **Open a spreadsheet application of your choice.**

☐ **Click once in the cell in the top left of the spreadsheet.**

☐ Paste the data.

Because we included a Tab key between the data elements, the data will align itself neatly into the spreadsheet, each variable taking up one cell. The Return key at the end assures that each person's data begins on a new row:



From here, you can use the spreadsheet to analyze the data.

Appendix A

Packaging

OK, you've developed a fully functioning Authorware file. Perhaps it is the Slide Show from Chapter 1, or the Space Shuttle Commander simulation/game from Chapter 4, or maybe it is one of your own original creations. Now you want to put your program to use. In other words, you want to install it at the site at which it was intended for you or others to use. What do you do?

As an experienced computer user, you know that all the computer files you create can only be opened with an appropriate application, almost always the same one used to create it (graphics and word processing files are obvious exceptions) The application must be either installed on the computer you happen to be using or accessible over a local area network. Authorware is no exception. All of the files you've created up to this point only open and run if Authorware has already been properly installed on your computer. If you copy your file to another computer that does not have Authorware installed, you will not be able to open or run it.

The computer will probably give you an error message of some sort indicating that the application is either "busy or missing."

So, should you just cross your fingers and hope that Authorware is already installed on all the computers you intend to use? Of course not. Not only is it highly unlikely that Authorware will be installed, it is also not preferable for end users to interact with the file in the same way in which we have created it. We don't want people to do things such as pause the program, jump to the icons, or make corrections (such as moving icons around the screen). In other words, we don't want users to have access to "author mode." Instead, we want them to run the program in "user mode," or a fixed version of the file that runs as programmed, but without all of the author features. We also want to be able to install the program on lots of computers without having to buy lots of copies of Authorware. The idea is to "author once, distribute many." The solution to all of these problems is a process called *packaging*.

If you bought one copy of Authorware, then you have the legal right to install it on only one computer.

What is packaging?

Up to this point, you have been creating and working with *unpackaged* files. Packaging is a process in which you make fully functioning files that are *not dependent* on the Authorware application to work. Instead, packaged files depend on another application — "RunA4M" on the Macintosh and "RunA4W" on Windows. RunA4M or RunA4W are often referred to as runtime applications. This application can also be embedded in the file itself, so that the file acts as a stand-alone piece — this is referred to as packaging *with* RunA4M or RunA4W. If you choose not to package with the runtime application (i.e. not embedding RunA4M or RunA4W in your file), you can make as many copies of this runtime application as you wish and install it on as many computers as you need. You should think of the Authorware application that we have used in this book as being used solely for the *authoring or development* of the file, and the runtime application as being used thereafter for the *distribution or implementation* of the file in final form. Packaged files cannot be changed or modified. Neither you or the user has access to the flow line or any of the authoring features and commands.

This assumes you are not planning on selling your program. Contact Macromedia for information on distributing the runtime software for commercial purposes.

It is important to understand that the outcome of packaging an Authorware file is *another* file. Do *not* think of packaged files as replacing the unpackaged files. I like to think of packaged files as being like the "children" to my original Authorware file. You can always create more children from the parent, but you cannot create the parent from the children! Likewise, you can always create another packaged file from the original unpackaged file, but you can never create an unpackaged file from a packaged one. If you ever want or need to make a change to the file, you must go back and change the unpackaged file (using the Authorware application), and then create a new packaged file. Therefore, it is critical that you save, protect, and backup the original unpackaged file.

Important! *Read this carefully and be sure you understand.*

How to package

Despite how long and involved the following steps may seem to you at first, rest assured that packaging really only takes a few minutes once you know what options you want. Fortunately, you can always package a second or third time until you finally get it the way you want. There are several decision points along the way. For example, do you want to package the file with the runtime application RunA4M or RunA4W embedded in the file or package it separately from the runtime application? If you are only distributing a single file, then it makes more sense to package *with* the runtime application. However, if you are distributing two or more files (such as all the projects you've completed in this book), then it makes more sense to package *without* the runtime application embedded in each file. Why? The runtime application can take up to 2.6MB of disk space. If you package

I use Authorware files in almost all of my courses. I am usually working on the original unpackaged file up to five minutes before class, then I quickly package the file and copy it to the computer in the classroom where I teach. The next day I do the same thing, replacing the previous day's packaged file with the current version. The unpackaged file is always safe and sound on my office computer (and backed-up).

Macintosh Alert!

The "fat" version of RunA4M is 2.6MB. The "ppc" version is 1.6MB and the "68K" version is 1.2MB. Older versions of the runtime application require less disk space.

a 100K file with the runtime application, you could have one file that is approaching 3MB in size! If you have five 100K files, you will need about 13.5MB of disk space if you package each with the runtime application versus about 3.1MB if you package the files without the runtime application. You still have to have one copy of the runtime application installed on the disk, but each of the five files can access it one at a time.

Five 100K files plus the 2.6MB runtime application.

I'm in a hurry, just give me the basics!

Here are "quickstart" instructions on how to package a file. Use the following as a quick way to try your hand at packaging or as a handy reference guide for later use. We'll describe each of the following steps in more detail in the next section.

Quickstart instructions on packaging a file (without libraries):

Libraries must be packaged in the same way as programs. The concept of a library is discussed in Appendix B.

1. **Select "File" under the "Modify" menu, then select "Properties..."; verify all of the file's options; click "OK" when done.**
2. **Run the file and make sure it works properly.**
3. **Select "Package..." under the "File" menu.**
4. **If packaging more than one file, choose to package "Without Runtime"; if packaging just one file, choose to package for the computer system you expect users to have;**
5. **Select the option "Include Fonts."**
6. **Click "Save File(s) & Package..."**
7. **Select the destination for the file and verify the name.**
8. **Click "Package."**
9. **Go to the desktop, double-click on the packaged file and be sure everything works properly.**

Platform Alert!

Macintosh options are "For Standard Macintosh," "For Power Macintosh Native," and "For all Macintosh Models."

Windows options are "For Windows 3.1," and "For Windows 95 and NT."

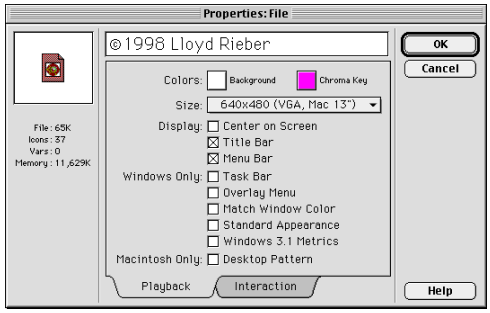
One more time, in slow motion

Now, let's take these steps one at a time and briefly consider the meaning (and various choices) of each step.

☐ Select "File" under the "Modify" menu, then select "Properties..."; verify all of the file's options; click "OK" when done.

Before packaging, it's a good idea to review all of the options associated with the file and make any necessary changes.

Here is the dialog box for the file's properties:



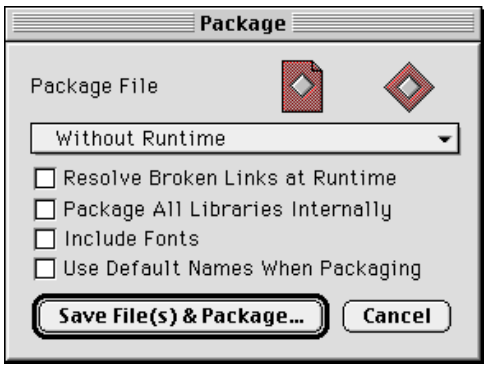
At this point, the most frequent option to review and possibly change is the file's internal title — whatever appears in the text window at the top will also appear in the Title Bar in the packaged piece (unless, of course, you choose to turn off the Title Bar). I usually put the file's final name here or a copyright notice (as shown above). It's also a good idea to verify that the program will "Restart on Return" (click on the Interaction tab to view this option). When this option is chosen, the program is assured to restart at the very beginning each time it is opened. In contrast, "Resume" means that the program will remember where the user was when they quit and will return automatically to that exact place the next time the file is opened. The "Resume" feature only makes sense for special applications, such as computer-based tutorials in which a user would start and stop frequently. Of course, "Resume" might be a useful feature if you have developing an adventure game that will take many sessions for the user to complete.

☐ Restart the file and make sure it works properly.

It is important to remember that packaging puts the program into a fixed form, mistakes and all. Any problems associated with the unpackaged file will be there in the packaged version. You'll save much time later by taking a little time now to go through your file thoroughly to correct all problems.

☐ Select "Package..." under the "File" menu.

The following dialog box opens:



Authorware 4.0 runs in "native mode" on Power Macintosh computers. In other words, this version takes full advantage of the RISC chip found in these computers. The end result being that Authorware files run very fast.

You have a variety of options at this point. The most important of which is whether or not to package the file together with the runtime application (RunA4M on Macintosh and RunA4W on Windows). As already discussed, the issue is really just one of disk space. Every file you choose to package with the runtime application will be up to 2.6MB larger than without. You have to have at least one copy of the runtime application on the computer anyhow, so if you are only going to package and distribute one file, you might as well go ahead and package with the runtime application.

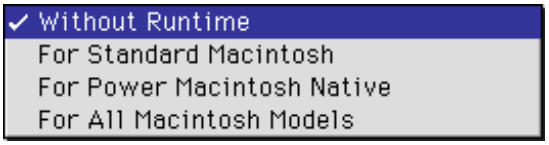
Of course, it might be easier for end users to install your program if RunA4M is embedded in the file since there are less files to deal with.

For simplicity's sake, I will be showing only the Macintosh options. Obviously, Windows users should make choices respective to their system.

☐ **If packaging just one file, choose to package "For Standard Macintosh," "For Power Macintosh Native," or "For all Macintosh Models" depending on which Macintosh you expect users to have; if packaging more than one file, choose to package "Without Runtime."**

Windows Alert!

Your options are "Without Runtime," "For Windows 3.1," and "For Windows 95 and NT."



☐ **Select the option "Include Fonts."**

Windows Alert!

This option is not available for Windows users.

Ever been in the situation where the computer substituted a system font (usually Geneva or Chicago) for one of your fonts? This happens when a custom font is not installed on the computer on which you are running the program (unpackaged or packaged). This typically happens when you are not using the *same* computer throughout the development of a single program (e.g. in a school's computer lab). It is not uncommon for different computers to have very different sets of fonts installed. If you began authoring the program on a computer with one set of fonts and finished it on another computer (with a different set of fonts), Authorware will substitute a system font for any text typed in a font not found on the second computer. If you carefully followed step number 1 and ran your file thoroughly making sure that it works properly before packaging and found that all fonts looked OK, then you can be assured that the proper fonts are installed on this computer. However, will end users have all these same fonts installed on their computers? By selecting the option "Include Fonts," Authorware will copy and paste the necessary fonts from your computer's system folder *directly into the packaged file*. Of course, this means

Font substitution is a very annoying and frustrating experience!

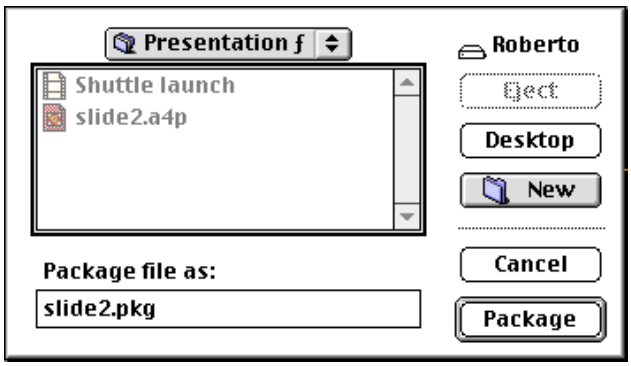
Fortunately, recent versions of Authorware do a good job of warning authors of font problems.

that the packaged file will be slightly larger than before, but in the vast majority of cases, this difference is marginal.

It is important to note that this option only works if you are using the standard TrueType or bitmapped fonts installed on most computers. If you are using other fonts, it is uncertain whether or not the fonts will be copied. (Try it and see!)

☐ Click "Save File(s) & Package..."

The standard file dialog box appears next:



☐ Select the destination for the file and verify the name.

Frankly, I prefer to have one folder for all my unpackaged files and another folder for the packaged files.

Be sure to give this file a name other than that of the unpackaged file if you are saving to the same folder. (Authorware will automatically attach either the file name extension ".a4r" or ".pkg" for this reason.)

☐ Click "Package."

Authorware first saves the current unpackaged file, then constructs and saves the packaged file. When done, the packaged file is saved to whatever destination you chose. Authorware then returns you to the flow line where you can continue working on the unpackaged file.

It's a good idea at this point to check to be sure the packaged file runs appropriately.

☐ Go to the desktop, double-click on the packaged file and be sure everything works properly.

Remember, if you packaged without embedding the runtime application into your file, you will need to give users a copy of the runtime application that matches the packaged version.

For example, Macintosh users would need one of the following (depending on how you packaged the file):

Windows users would need the application “RunA4W.”

Depending on the features you include in your file, you will probably need to also give end users a folder called “Xtras.” This should also be placed in the same folder or directory as the runtime application (or the file itself, if you embedded the runtime application in the file itself).

The concept of Xtras was introduced with Authorware 3.5 as a way to easily add tools to Authorware menus and extend Authorware's functionality (it’s useful to think of them as a kind of plug-in). Many Xtras are available at the Macromedia web site free of charge (www.macromedia.com).

Appendix B

Libraries

Libraries provide you with an easy way to reuse parts of your Authorware file in an efficient manner. Libraries are stand-alone files that can be shared with many Authorware programs. Like regular Authorware files, libraries contain icons, though they are limited to the following: Display, Interaction, Digital Movie, Sound, and Calculation. Each Authorware file can access a total of seven different libraries.

There are two good reasons for using a library: management and economy. Once you become proficient with using libraries, you will find it much easier to organize and plan multimedia programs. However, most people are initially drawn to libraries because they offer a practical means of keeping their Authorware files down to a reasonable size. As you know graphics, animation, and sound can be very memory intensive. If you have ever scanned a color photograph, you know that a single display can easily require one megabyte of memory for storage. If you wanted to include such a scanned image in your file five or six times, you would have to copy and paste the Display icon that contains it that many times (increasing your file by 1MB each time). Libraries offer you with an easy way to manage these memory-hungry multimedia units so as to minimize the amount of memory your file will ultimately need. While you might think of a programming solution to problems such as these (such as including a single Display icon inside a loop that successively gets displayed then erased), libraries present a simpler (and usually preferred) approach. Using libraries is a must for anyone using Authorware to develop multimedia.

Why haven't we used libraries before? Frankly, we just haven't needed them to build the prototype projects in this book. If you enhance the overall design of these projects or your own (by adding improved graphics and more sound), you will find libraries a big help.

As another example, consider a game that uses four different sounds where each corresponding Sound icon requires 25K of memory. If each Sound icon was used just once, the file would need an additional 100K for all the sounds. However, imagine that the game uses each sound dozens or even hundreds of times. The temptation is just to copy and paste the Sound icon in each place we need it in the program. If we do this, it's not hard to imagine the file growing to an enormous size. The solution is to create a library where the four Sound icons are *really* stored and then put links or pointers in your Authorware file that tell Authorware to go play the sound in the library. Since libraries are stand-alone files that can be shared by many Authorware programs, you

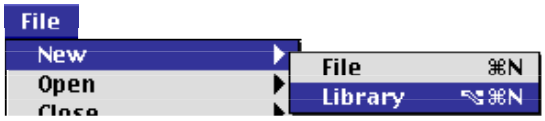
could create separate libraries for sound effects, backgrounds, animations, etc.

Creating and using a library

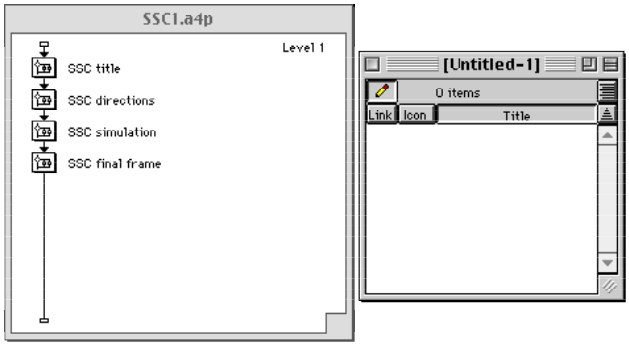
While authoring an unpackaged file, each open library appears in a separate window on the screen. You drag and drop icons into a library's window just as you would a regular Authorware file. When you want to use an icon stored in a library, you simply drag the icon from the library's window to the Authorware file's flow line. This does *not* copy the library's icon, but merely creates a link to it, very similar to the idea of an alias. Let's quickly go through the process of creating and using a library.

There are differences between an alias and an icon linked to a library, but it's a very good analogy for beginning to understand how a library works (assuming, of course, you've used aliases).

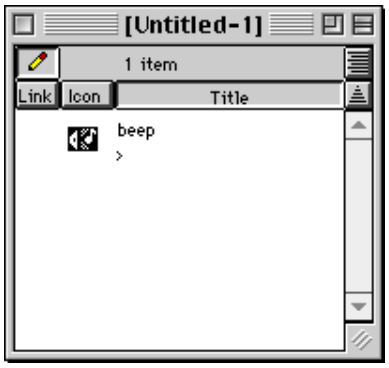
To create a new library, select "New" from the "File" menu, then select "Library" (or select "Open" from the "File", then "Library", to open an already existing library):



When you do, a new untitled window opens on the screen:



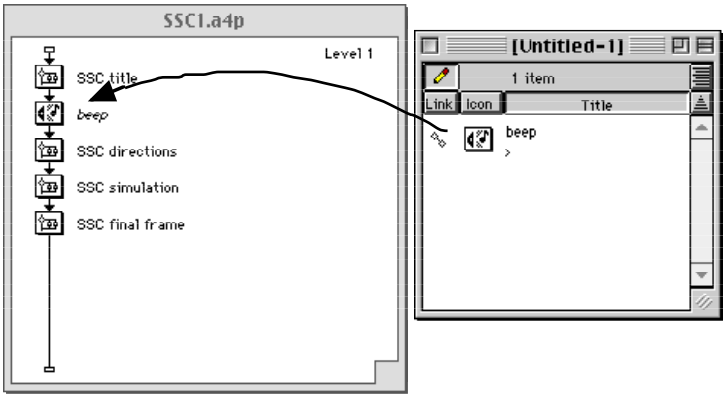
To add an icon to the library, just drag and drop an icon into the library's window. For example, drag a Sound icon to this window and title it "beep."



You can now work with this icon just as you would normally (such as by double-clicking on the icon to open it and then defining what the icon will do).

To use the icon in your file, just drag and drop it from the library window to the flow line in your Authorware file:

You can do this as often as you wish. You now have an "endless well" of beep icons. Each icon you drag and drop from a library consumes just a tiny fraction of memory.



Notice that the title of the icon in the Authorware file's window is italicized. This signifies that this icon is simply a link to the *real* icon contained in a library. Any change you make to library icons will likewise be made in all the linked icons. You can double-click on the italicized icon to open it (if you do, notice that the title in the dialog box shows the icon is part of a library), however, you will be very limited in the kinds of changes you can make (e.g. the position of Display icons can be changed in the linked icon without changing the original).

You save libraries just as you would any other file. Make the library window active (by clicking once on it) and select "Save As..." from the "File" menu. Authorware will remind you to save changes to a library when you save or close your Authorware program.

When packaging a file, you can choose to package the library as a separate file (useful when many packaged files are accessing the library) or you can embed the library directly into the packaged file.

You do so by choosing the packaging option "Package All Libraries Internally" (see the illustration on p. 360).

Appendix C

Jumping out to other files and applications

As you do more development with Authorware, there will be many times when you will want to go or jump from one file to another. Authorware provides you with a variety of "jump" functions to do just that. The function "JumpFile" lets you jump seamlessly from one Authorware file to another. The function "JumpFileReturn" does the same thing, except that it returns automatically to the "jumping off point" in the original file when the user chooses to quit the second file. The only catch to the "JumpFileReturn" function is that you have to have enough RAM available for Authorware to hold all the files in memory. In addition, you can pass variables back and forth between files with these functions, thus allowing for any data collected in one file to be used by additional files.

The JumpFileReturn function can be nested within succeeding files. You can jump from the second file to a third file, and then to a fourth file, etc. When returning, Authorware takes the same "path" in reverse order going from the fourth file to the third, then the second, and finally back to the first.

Here is the "official" syntax for both functions:

```
JumpFile("filename"[, "variable1, variable2, ...",["folder"]])
JumpFileReturn("filename"[, "variable1, variable2, ...",["folder"]])
```

All elements inside the brackets (i.e. the names of any variables you want to send to the second file and the pathway to a file not in the same folder) are optional (brackets would not actually be typed).

Consult the Authorware manuals for the details on how to set up pathways to files located in other folders or directories.

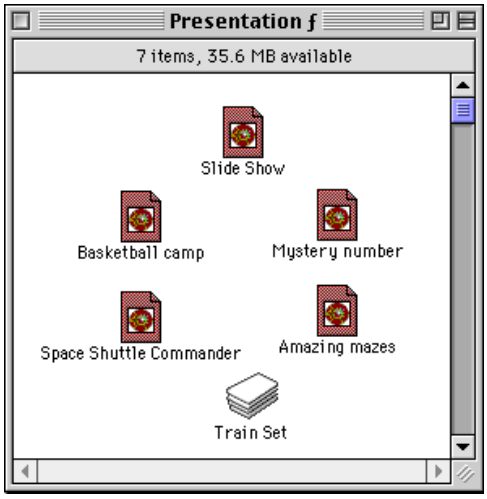
Not only can you jump from one Authorware file to another, but you can also jump to files created in other applications with the functions "JumpOut" and "JumpOutReturn." These work in a way almost identical to the other two jump functions except that you also must identify the application to which the file belongs. Again, the catch with "JumpOutReturn" is that you must have enough RAM to allow all applications to be loaded and stored in memory throughout the execution of their files. Obviously, any applications identified in a "JumpOutReturn" must be loaded and available on the computer's hard drive or on a local area network.

Here is the "official" syntax for both of these functions:

```
JumpOut("program"[, "document", "creator type"])
JumpOutReturn("program"[, "document", "creator type"])
```

For example, suppose we wanted to include the games and simulation projects completed in chapters 2-5 as examples in the slide show we created in chapter 1. While it is possible to "copy and paste" each project into the slide show, a more practical approach is to jump back and forth from the slide show and the projects. You might also find other suitable examples created in other applications. For example, an interesting simulation/game that comes with HyperCard called "Train Set" would also make a great example to include in the slide show. Let's elaborate on this example to show how to use Authorware's "jump" functions.

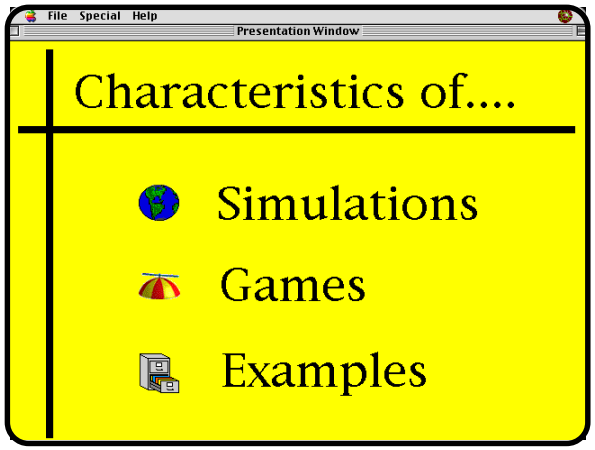
While it is possible to jump to files contained in other folders or directories, the simplest approach is to put all of the files together in the same folder. For example, I've put all the unpackaged files completed in the first five chapters plus a HyperCard stack "Train Set" in a folder called "Presentation *f*":



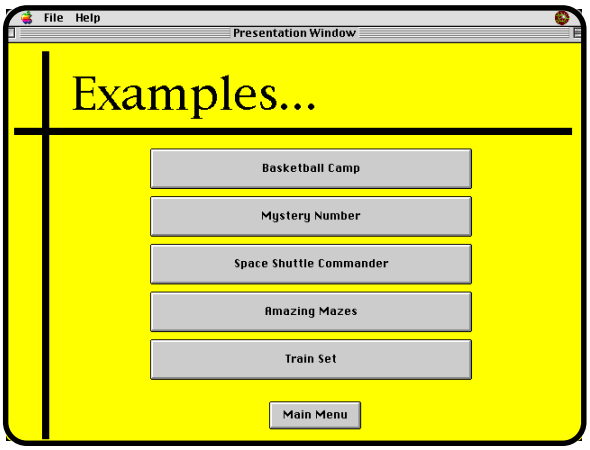
Obviously, I renamed the files according to my own preferences.

Be sure that it is legal for you to make a copy of any files or applications you did not create.

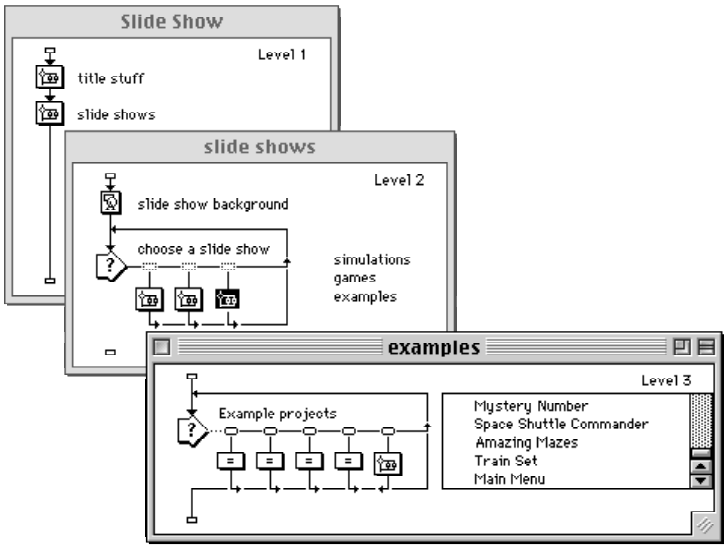
I modified the Slide Show file to include another choice called "Examples" on the main menu:



When “Examples” is chosen, a secondary menu is displayed showing the five examples plus a button to go back to the main menu:

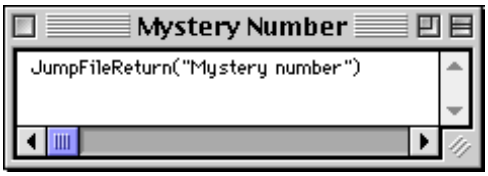


Here is the corresponding flow line for the modification to the main menu and creation of the secondary menu:



As you can see, I added a Calculation icon for each of the five examples (you would have to scroll to the top to see “Basketball Camp”). Since I want to jump to each example and then return back to this secondary menu, I need to use the “JumpFileReturn” function to run each of the Authorware files and the “JumpOutReturn” function for to launch the HyperCard stack “Train Set.”

Here's the programming to jump to and return from the Authorware file "Mystery Number":



Since no pathway is specified, this file must be in the same folder as "Slide Show". It is also critical that the actual file name exactly matches what you type here.

This is the simplest example possible since the two files are contained in the same folder and there are no variables being passing back and forth between the files. When this command is executed, the file "Slide Show" stays in memory while the file "Mystery Number" is loaded into memory and run. When the user chooses to quit "Mystery Number" Authorware returns to the secondary menu in "Slide Show". You would need to program the corresponding code for the other four Authorware files.

Likewise, here is the programming to jump out to the HyperCard stack "Train Set":

If you were jumping out to a stand-alone application, you would omit the file name information.

Again, since no pathway is specified, this file must be in the same folder as "Slide Show" (although the application does not). Likewise, it is critical that the actual titles of the application and file exactly match what you type here.

When this command is executed, the file "Slide Show" stays in memory while the application "HyperCard" is launched, followed by the loading of the stack "Train Set." When the user chooses to quit the application "HyperCard" it is removed from memory and the user is returned to the secondary menu in "Slide Show".

You can test out the functions while the files are unpackaged. Package the files to another folder and keep the same names for the files. Also move (or copy) the "Train Set" stack to the folder containing the packaged files.

***Warning**! If you use the same names for both the unpackaged and packaged files, be careful you don't overwrite your unpackaged files with the packaged files.*

Appendix D

Other features and issues you need to know

While this book has never pretended to try to show you everything there is to know about Authorware, there are still a few more features and issues with which you should be familiar. This appendix provides a brief summary of compacting files, printing files, memory issues, and issues surrounding cross-platform development. One other important feature is reserved for the next Appendix — how to "shock" your files in order to them run over the internet and intranets. These are meant as a quick and practical overview and not as a substitute for studying the details in the Authorware manuals.

Compacting files

First of all, compacting is not the same as compressing or compiling. Compacting is similar to the process of defragmentation available through a variety of system utility applications, such as Norton Utilities. The purpose of compacting is to optimize the space and speed of Authorware files. As you work with a file over time, copying and pasting icons, deleting icons, moving icons, etc., the area of the disk that holds the file becomes fragmented, like holes in Swiss cheese. The process of compacting rewrites the entire file from scratch, reviewing and reconstructing each icon in the file separately. To compact a file, select "Save and Compact..." from the "File" menu. Because it rewrites the entire file, this option takes a little longer to save than the typical "Save" option.

There is very little written about this process or feature in the Authorware manuals.

There are no hard and fast rules for when to compact. In a sense, you don't really have to do it at all. However, as files grow you may find that compacting occasionally will reclaim a great deal of unused disk space reducing the file size significantly. As already mentioned, this will also increase the access speed to parts of the file from the disk, thus increasing the file's overall performance (especially when accessing digital movies). When you package a file, the packaged file is automatically compacted.

Although the Authorware manuals imply that you can use "Save and Compact..." to rewrite a file with the same name, I have not found this to be the case. As a result, I suggest the following procedure when you want to compact:

1. At the desktop, backup the original file;
2. At the desktop, change the name of the original file to something like "title unc.a4p";
3. Open the renamed original file.
4. Select "Save and Compact..." from the "File" menu and choose to save the file with the original name (e.g. "title.a4p").
5. Continue to work with the compacted file from there on (until you choose to compact the file again).
6. Discard the renamed original file.

In other words, discard "title unc.a4p".

Printing files

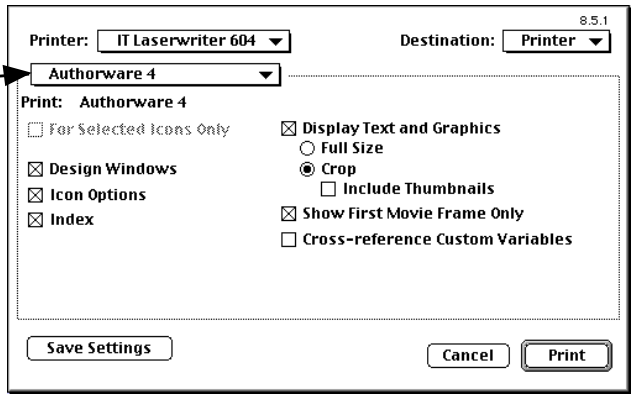
Authorware has excellent printing features to put all aspects of your Authorware file on paper (e.g. its program logic as shown by the flow lines, screen displays, contents of Calculation icons, etc.). Because a printed copy safely holds the *intellectual* property of a file it acts as one of your most important "last resort" backups. You can wave magnets around paper, spill coffee on it, even cut off the corners and still make perfect sense out of the printed copy. One big drawback to printing is that it consumes an amazing amount of paper to print out even a small file when you have all the printing features turned on. Therefore, I recommend that you approach printing cautiously and monitor the printing process carefully each time you select it. Otherwise, you may find yourself with an unexpected 200-300 page printout!

Of course, it's no small chore to reconstruct the file from scratch solely from the printed copy, so be sure to keep lots of electronic backups on hand as well!

To print all or part of your file, select "Print..." from the "File" menu. When you do, the following dialog box appears:

Also take note of the "Print Screen" choice under the "File" menu. This is an excellent way to quickly print out a "snap shot" of the screen to examine the current flow line or edit some screen display.

Be sure you are viewing the Authorware 4 options.



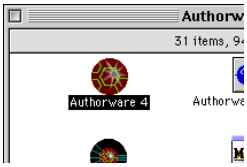
Most of the options are intuitive, but consult the Authorware manuals for the details. Again, be careful that you don't wind up with a ream of paper unnecessarily. (A good approach at first is to choose to print in groups of 10 or 20 pages just to make sure you are getting what you expect.)

If you take the recommended route of officially registering your copyright on your original files, you will need to print out a portion of your file to accompany the application. See Appendix F for more information on copyright.

Memory issues (Macintosh Only)

Memory problems are among the most confusing and mysterious problems to troubleshoot. Fortunately, most of the problems can be resolved with a few simple tricks. For example, if you find your program crashes due to a Type I error, it is almost certainly due to not enough RAM being allocated to the Authorware application (for unpackaged files) or the RunA4M application (for packaged files). Assuming your computer has enough RAM, it is easy to increase the amount of RAM partitioned to any application on the Macintosh. Of course, don't forget to consider how much RAM the typical end user would have on their computer. It is easy to forget that the average user typically has a computer with less "horsepower" than those used by most developers.

To determine how much RAM your computer has, select "About this Macintosh..." under the Rainbow apple menu when at the desktop.

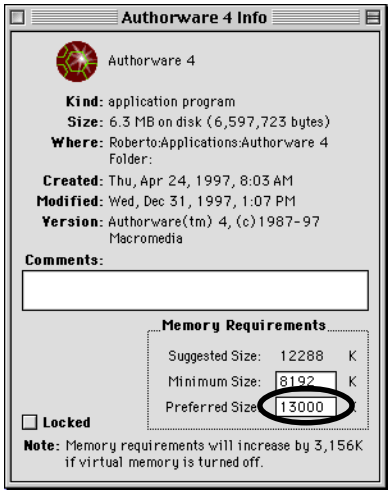


For example, to increase the amount of RAM allocated to the Authorware application, click once on the application's icon when at the desktop to highlight it.

The application cannot be in RAM (i.e. running) when you do this.

Next, select "Get Info" from the "File" menu. The following dialog box appears:

If you don't see "Get Info" you are probably not at the desktop, but inside some other application.



Increase the "Preferred size" to an appropriate amount. The optimum amount will be based on a trade-off of what it takes to get the file to run smoothly versus what you expect end users to have available on their computers.

If your unpackaged piece runs fine but your packaged file does not, it's probably because its application (RunA4M if packaged separately or the file itself if you chose to embed the runtime application in the file) does not have enough memory allocated to it.

Other memory issues to consider

In general, the amount of memory required to run an Authorware file is always the largest amount of memory required at any point in the file. Many programs will run sluggishly (or crash) when they reach a point where a memory intensive chunk is located, such as a scanned color graphic or a sound file. Your files will run better and take up less space if you carefully consider ways to minimize such memory-hungry chunks. One trick is to use a graphics application (such as PhotoShop) to break one large graphic into four blocks which are placed carefully side by side in one Display icon. Although it will look like one graphic to the end user, the Authorware (or RunA4M) application will only have to process one fourth of the graphic at a time, thus increasing speed and reducing the amount of memory needed.

Of course, you should also consider embedding 8-bit graphics (requiring only 256 colors) instead of 24-bit graphics whenever possible. Many times, end users will only have a monitor capable of displaying a 8-bit graphic even though the computer still needs to process the 24-bit graphic.

Finally, you should review the "Preload" function. As the name implies, this function allows you to preload certain kinds of information so that they are "ready and waiting" in RAM when you need them.

Cross-platform development

Cross-platform compatibility (Macintosh to/from Windows) has been one of the goals of Authorware from its inception. However, given the differences between the hardware and operating systems of these two platforms, such development is far from seamless. If your goal is to author something that will used on either the Macintosh or Windows platform, you should carefully consider the needs of both platforms at the start and stop often to test the piece on the platform other than the one on which you are developing. Regularly checking how graphics, fonts, sound, movies, etc. appear on each platform as you author a piece can help to avoid time consuming and costly problems later.

In theory, porting either a Macintosh or Windows Authorware file to the other platform is easy. Starting with version 3.0, you can open any Authorware file with either the Macintosh or Windows version of the Authorware application (assuming that the file is saved to a disk that the computer can read; Macintosh users should be careful to save their file on a disk formatted for Windows).

One of the significant enhancements to verson 4.0 is the ability to use a packaged file on either a Macintosh or Windows computer, assuming that the appropriate runtime software is installed.

Display sizes and colors

The screen size in which an Authorware file's display window is shown can vary a great deal from computer to computer. Getting true colors on both platforms is also tricky. Authorware does it best to match colors across platforms. However, if you author a piece with 256 colors but run it on a computer with only 16 colors, the results can be very disappointing. Again, the best recommendation is to check the status of the project on both platforms when you start authoring and then continue to do so throughout the development process.

Fonts

Be careful in which fonts you choose to use. Not all fonts are available on both platforms and not all fonts look the same on both platforms. Experiment with fonts and font sizes at the start to find the right balance. Fortunately, Authorware gives you some control for specifying which fonts to substitute for a missing font. Once you choose fonts that look good on the respective platform, you can select "File" from the "Modify" menu, then select "Font Mapping" to set up a font map.

There are many other things to consider as well, such as making sure you have the right Windows drivers for your display and QuickTime movies. Other problems can arise due to the different conventions in naming files, folders, and pathways. Many Windows computers still use the "8.3 format" for naming files (eight characters maximum for the file's name followed by an optional three-character extension with a period in-between).

Appendix E

Shockwave: Preparing Authorware files for use on the world wide web

One of the most recent and dramatic enhancements to Authorware is its compatibility with Macromedia's Shockwave technology. Beginning with version 3.5, packaged Authorware files can be converted (or "shocked") into a form that can be downloaded and run on the World Wide Web and intranets. Without Shockwave, the only alternative for distribution is compressing a packaged Authorware file (along with the runtime application) and allowing it to be downloaded from a web or FTP site. After downloading, the files would have to be uncompressed into their original size and form. Since the files are then saved to the user's hard drive, they are then opened and run separately from the web browser application. At its core, Shockwave is a compression utility that reduces and segments files to enable them to be transmitted over networks much more quickly than downloading intact files. Shocked files are also viewed with a Shockwave-compatible browser, such as Netscape or Internet Explorer, with the Shockwave software (called a plug-in) installed. Furthermore, shocked files can take advantage of several web-related functions allowing the shocked file to link to and interact with other web addresses. The result of this being that the Authorware files become an integral and seamless part of the web experience. Authorware files run virtually intact when shocked. This means that if you know Authorware, you also have command of a powerful authoring tool for internet applications.

Intranets are wide area networks owned and operated by organizations (such as businesses). These act like "mini-world wide webs" for the organization's internal use. They may or may not be linked to the Internet (if so, intranets are usually "shielded" from intruders with special software called a "firewall").

This appendix is meant to give you a brief overview of the entire process and assumes you have experience in using and developing HTML files. All of the complete information and resources you need to shock files can be found or downloaded from Macromedia's web site (http://www.macromedia.com). This appendix should give you the gist of the process.

There are five steps to this process: 1) downloading and installing the Shockwave plug-in; 2) converting a packaged Authorware file (without runtime) with the Afterburner application; 3) modifying your web server software to enable shock files to be downloaded by people accessing your

web site; 4) writing a HTML document that embeds a link to the shocked files; and 5) copying all of the necessary shockwave files, resources, and HTML documents to your web server.

Step 1. Downloading Shockwave Plug-in software from the Macromedia web site

The first step is to download the necessary software from the Macromedia web site. This is also a great place to get reference materials, background information, sample files, resource files, etc. However, the Macromedia web site is very large making it difficult at times to find the right resources quickly. Therefore, I've provided some web addresses below to get you to the resources you need now. Of course, web sites have a habit of changing, so if any of the following addresses don't seem to work, I advise you to start with Macromedia's home page (www.macromedia.com) and start searching for these resources from there. However, as of this writing, Macromedia has the following web site where you will find many useful development tools, resources, and information:

http://www.macromedia.com/support/authorware/how/shock/

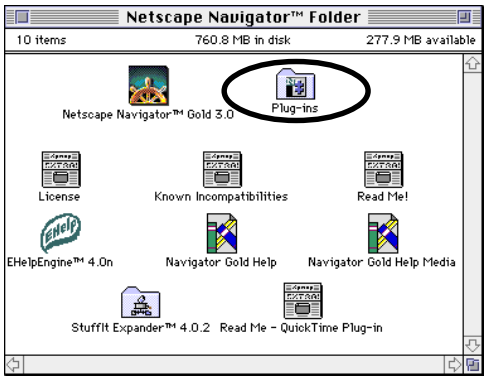
You (and your end users) will also need to get the Shockwave for Authorware plug-in for your web browser. Again, the Shockwave plug-in only works with compatible browsers (such as Netscape and Internet Explorer).

The Shockwave for Authorware plug-in can be downloaded from this web address:

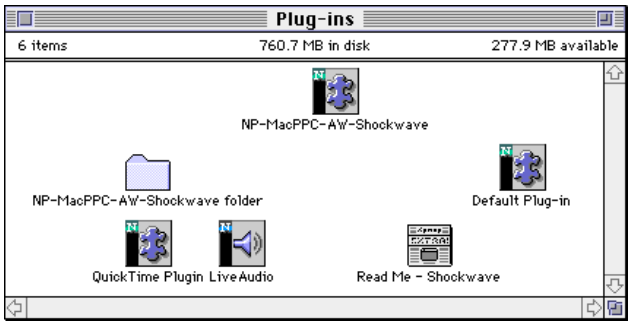
http://www.macromedia.com/shockwave/download/alternates/index.html

Be sure to download the correct plug-in for Authorware. Macromedia provides Shockwave plug-ins for their other products as well and it is easy to get confused. A good recommendation is to download "Shockwave - The Works" — this will give you all the plug-ins you need to run all of Macromedia's shockwave compatible software.

You will need to install the Shockwave plug-in in the Plug-Ins folder for your web browser:



After installation, the plug-ins folder will contain at least the following files:



Of course, you might have other plug-ins installed as well.

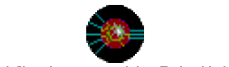
Step 2. Shocking an Authorware piece

The next step is to shock a packaged Authorware file. If you haven't done so already, you need to package an Authorware file without the runtime application. (The runtime software is included in the plug-in software.)

Next you need to use the Afterburner application to compress and segment your packaged file into manageable pieces, called segments, in order to facilitate their transmission over the network. The result of this process are two kinds of files: one map file and one or more segment files. The map file contains information the Authorware plug-in will need to retrieve each segment and to retrieve any external files that your piece uses (such as movies). The default size for a segment file is 16000 bytes, though this can be changed.

This application comes with Authorware 4.0.

Here are abbreviated instructions for using the Afterburner application:

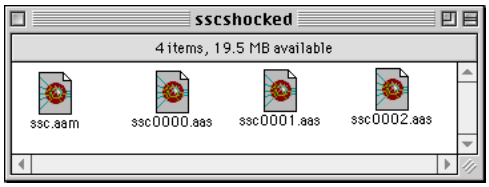


1. Open the Afterburner application and select the packaged file you wish to shock.
2. Select a destination folder where you want the resulting map file and all segment files to be placed.
3. In the Segment Settings dialog box, enter a prefix (up to four characters) that will be used to name the segment files (Afterburner will automatically number the segments using this prefix).
4. Enter a value for the segment size (keeping the default size of 16000 bytes is a good idea).
5. Afterburner then compresses and segments your Authorware file. The map file will open on the screen (this can be edited). Macintosh users will need to "flatten" any external files. One exception are QuickTime movies (assuming they have been saved to be playable on non-Apple computers; these also must have the ".mov" extension added to their titles).
6. Quit Afterburner.

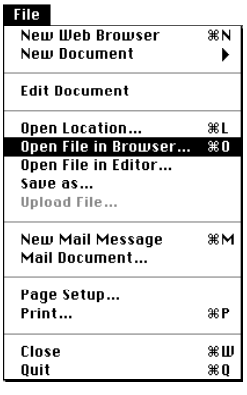
Windows Alert!

You will have a similar application.

For example, after shocking a packaged file titled "ssc" (a version of Space Shuttle Commander), the following map file and three segment files were saved to disk:



It's a good idea to test your shocked file right away with your web browser. This assumes, of course, that it is a Shockwave-compatible browser and already has the necessary plug-ins installed. For example, if using Netscape, you would open the application, then choose "Open File in Browser..." from the "File" menu, then select the map file you wish to run. The plug-in software takes over from there.



Step 3. Modifying your web server software

Before you can load your shocked files on the world wide web for users to access, you need to configure your web server to recognize and handle shocked media. It's highly recommended that only the web server's system administrator do this procedure. All of the information needed for this operation can be found at the following web address:

http://www.macromedia.com/support/authorware/how/shock/

For example, the server's MIME types file must be updated to include the appropriate MIME type lines.

Here are the Authorware MIME Type Mappings:

```
AddType application/x-authorware-map aam
AddType application/x-authorware-seg aas
AddType application/x-authorware-bin aab
```

Step 4. Writing the HTML file

A shocked file can only be accessed by a user over the world wide web via a HTML document that includes the EMBED command. For example, here is a simple HTML document that runs the "ssc.aam" map file produced earlier:

```
<HTML>
<HEAD>
<TITLE>SSC Shocked</TITLE>
</HEAD>
<BODY>
<EMBED SRC="ssc.aam" WIDTH=520 HEIGHT=320
WINDOW=onTop>
</BODY>
</HTML>
```

The width and height must match the dimensions of your screen display. The Window type can either be "onTop" or "onTopMinimize" (Windows users can also use the type "inPlace" — this runs the shocked file within the browser's window). There are a variety of other commands you can use, depending if you used the Macintosh or Windows version of Authorware. For example, the following three functions can be added to your Authorware files before packaging to enable your files, once they are shocked, to access other web sites:

```
GoToNetPage
NetDownload
NetPreload
```

Consult with the Authorware manuals and the Macromedia web site for more information on how to use these functions.

Step 5. Copying all of the files to the web server

The final step is to simply to copy all of the shocked files to a properly configured web server and to give users the web address (i.e. URL) to your site.

Appendix F

Copyright issues

You need to be concerned with copyright issues from the moment you begin programming. You have worked hard to produce creative programs and you need to protect your rights as would any author. Regardless of whether you intend to sell your programs or just give them away, you still need to remain in control of how the files are distributed. Here are some unofficial and informal recommendations and pointers related to copyright from one author to another. Obviously, all of my experiences have been in the USA and all of the comments below apply directly to American authors. I do not know to what extent they apply to other countries, however, copyright has long been an international affair in publishing. The increase of published materials on the internet (like this book) make international agreement of copyright protection crucial.

I am not a copyright lawyer and I cannot offer you advice as would a practicing attorney. If you are developing a "big time" commercial program, you might consider getting expert legal counsel.

Copyright exists upon the *moment of creation of a work in fixed form*. You do not technically have to do anything to possess a copyright. However, you should protect yourself from persons infringing your copyright (i.e. stealing) and also from inadvertently having your material become part of the public domain. Once in the public domain, *no one* can claim copyright, not even you. Proving that you own the copyright to a piece can be difficult unless you take some simple precautions. For example, you should display a notice of copyright in your program in a conspicuous place, such as the title screen, as soon as begin development. A display such as "Copyright 1997 Your Name" is sufficient. It's also highly recommended that you register your copyright with the Copyright Office in Washington, DC. It is relatively simple and inexpensive to do (unlike patents).

You cannot claim copyright for an idea.

You are not boasting when you display the copyright notice!

To register a copyright, you need to send the following three elements to the Copyright Office:

1. A completed application;
2. A nonrefundable filing fee of $20.00; and
3. A nonreturnable deposit of the work.

The application and "deposit of the work" varies with the kind of work. Computer programs (such as Authorware files)

usually need Form TX and the first 25 and last 25 pages of the source code (see "printing files" earlier in this appendix).

The application forms and instructions, as well as circulars providing background information on the copyright process, can be requested free of charge directly from the Copyright Office:

> Copyright Office
> Library of Congress
> Washington DC 20559-6000
> Regular telephone: (202) 707-3000
> Forms hotline: (202) 707-9100
> http://lcweb.loc.gov/copyright/

Almost all of this information can be downloaded from the Copyright Office's web page or you can request it by mail or phone. (The regular phone number is usually busy, but you can easily request specific forms via the "forms hotline." The web address is the best route to getting forms and information quickly.) At the very least, be sure to get the following forms and circulars:

- Form TX
- Circular 1 Copyright Basics
- Circular 7d Mandatory Deposit of copies or phonorecords for the Library of Congress
- Circular 61 Copyright Registration for Computer Programs
- Circular 93 Highlights of U.S. Adherence to the Berne Convention

Index