# Learning MicroWorlds Pro™

LCSI®

# Learning MicroWorlds Pro

by Tom Lough

## Acknowledgments

Many people have contributed to the development of this book. I would like to thank especially Sharnee Chait and Brian Silverman of LCSI and the following faculty members of Murray State University: Ann Assad, Clarance Benes, Kevin Dupre, Dwayne Green, and Martin Jacobs. Thanks to Kyser Lough, to Ms. Angie Murdock and her students at Calloway County Middle School, and to the middle school students in the MSU Super Saturday Program for their assistance. I also appreciated the comments of the various reviewers, including Natalie Rusk, Mark Guzdial, and Michael Tempel.

## About the Author

Tom Lough is an assistant professor of science education at Murray State University in Murray, Kentucky. Prior to that, he has been an engineer, a teacher of science and mathematics at both the high school and college level, and a product developer for LEGO Dacta, the educational division of the LEGO Group. He is the founding editor of Logo Exchange, an educational periodical dedicated to the Logo computer language, and has been actively involved with Logo-related activities since 1982.

Graphic design by *Le groupe Flexidée*



Legal deposit, second semester 2000

ISBN 2-89371-492-7
Printed 4-00

# TABLE OF CONTENTS

# *Preface*

**WELCOME TO MICROWORLDS PRO! One of the first questions to be asked about any software application is, "What can I do with this?" MicroWorlds Pro (short for Professional) was developed to help you deepen your understanding of MicroWorlds and to get a sense of the depth and breadth of what is possible in this multimedia programming environment. Everything that you learn here about MicroWorlds Pro can be applied to the MicroWorlds application currently used in schools around the world.**

**In order to best help you to *do* what *you* want with MicroWorlds Pro, I have put together a diverse collection of projects designed to introduce you to the versatility of the MicroWorlds Pro environment. Each chapter focuses on one project, and is followed by a "chapterette" containing additional ideas and extensions.**

**Work on the projects in any order you wish. (Chapter 1 contains the basic information you will need to start a project, and can be used for reference if you choose to start later on in the manual.) Follow your interests! As you do, you will be learning more about the power of MicroWorlds Pro and its underlying programming concepts. I hope that you will also be finding your own answers to what you can do with this exciting application.**

**The first chapter contains a presentation project. Here you will be introduced to some basic multimedia techniques and get a jump start on using MicroWorlds Pro to communicate your ideas effectively. By constructing a few pages yourself, you will get an idea of how quick and easy it is to use this multimedia application. Chapterette A contains additional hints and tips for animation and other special effects to jazz up your projects.**

**Interaction is the theme of the second chapter. I have selected an interactive game to help you learn more about programming and about project development as well. The chapterette tells you how to maximize interaction by publishing your project on the World Wide Web!**

**The final chapter focuses on using MicroWorlds Pro for research. A project in genetics was selected that is both straightforward and intriguing, and that can be used as a model for your own projects. You will learn some advanced programming techniques as well as some ways to display the results of your investigations. The chapterette shows how you can plot the results of other kinds of investigations and measurements.**

**As you work your way through these projects, you will begin to realize why MicroWorlds Pro has been called one of the most powerful and flexible multimedia environments available. Based on Logo, a computer language used in schools and universities around the world, MicroWorlds Pro can take you in directions you may never have dreamed possible.**

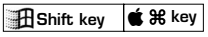
**Now turn the page and let the journey begin!**

**Tom Lough**
**Murray, KY**

**This manual is for use with both the Windows and Macintosh versions of MicroWorlds Pro.**

**The following symbols are used throughout this book:**


**With Windows, Right Click is used to open pop up menus on objects; the same functionality is obtained on a Mac by Control Click.**




**With Windows, the Shift key is used to add more than one shape to a turtle and to change the heading of the turtle; the same functionality is obtained on a Mac with the Command key.**

CHAPTER 1

# *Presenting . . .*

## Overview and Objectives

**THIS CHAPTER introduces some multimedia techniques and gives you a jump start on using MicroWorlds Pro to communicate your ideas. Your first project will be a presentation, given by a teacher at the beginning of a course. As you work your way through the chapter, you will:**

- **Explore a sample MicroWorlds Pro presentation project**
- **Create three pages as your own first MicroWorlds Pro project**
- **Learn many MicroWorlds Pro features and tools**

## Welcome . . . A Sample Project

MicroWorlds Pro files are called *projects*. With MicroWorlds Pro, you will be able to develop many different kinds of projects including presentations, interactions, and investigations or research projects. To get an idea of the potential of MicroWorlds Pro, let's begin by exploring a sample presentation project.

Start up the MicroWorlds Pro software. From the File menu, select Open Project. Open the Welcome project in the Samples folder.


Welcome is a typical presentation project, with a number of *pages*. Many of the objects in the project are programmed to do something. As you go through the project, click on various objects to see what they do.

**Note**

Plenty of help with MicroWorlds Pro is available if you need it.

- Help menu. The Help menu at the top of the screen includes Help Topics (the on-line Reference Manual), Vocabulary (definitions and examples of all the words in MicroWorlds Pro), and information on recent error messages.
- Project tree. The Project tab on the right side of the screen displays a helpful tree diagram of the various objects in your project and the relationships between them.
- Printed material. Refer to *Tips and Tricks* for more information.
- Online help. You can send your questions by e-mail to help@lcsi.ca

## STARTING FROM SCRATCH

Now it's time to try your hand with MicroWorlds Pro! Imagine the following scenario. You and I are preparing to co-teach a science methods class at Murray State University here in my home town of Murray, Kentucky. We are preparing for the first class meeting, and decide we want to use a multimedia presentation to emphasize some course information. We also want to stimulate our students' interest in technology and demonstrate that technology can be used in a variety of ways. At the same time, you want to learn more about MicroWorlds Pro and multimedia techniques. We decide that you will develop several presentation pages and that I will help you along the way. Once you gain some experience with MicroWorlds Pro, you will be able to do even more.

Let's get started on the first project page and learn how to:

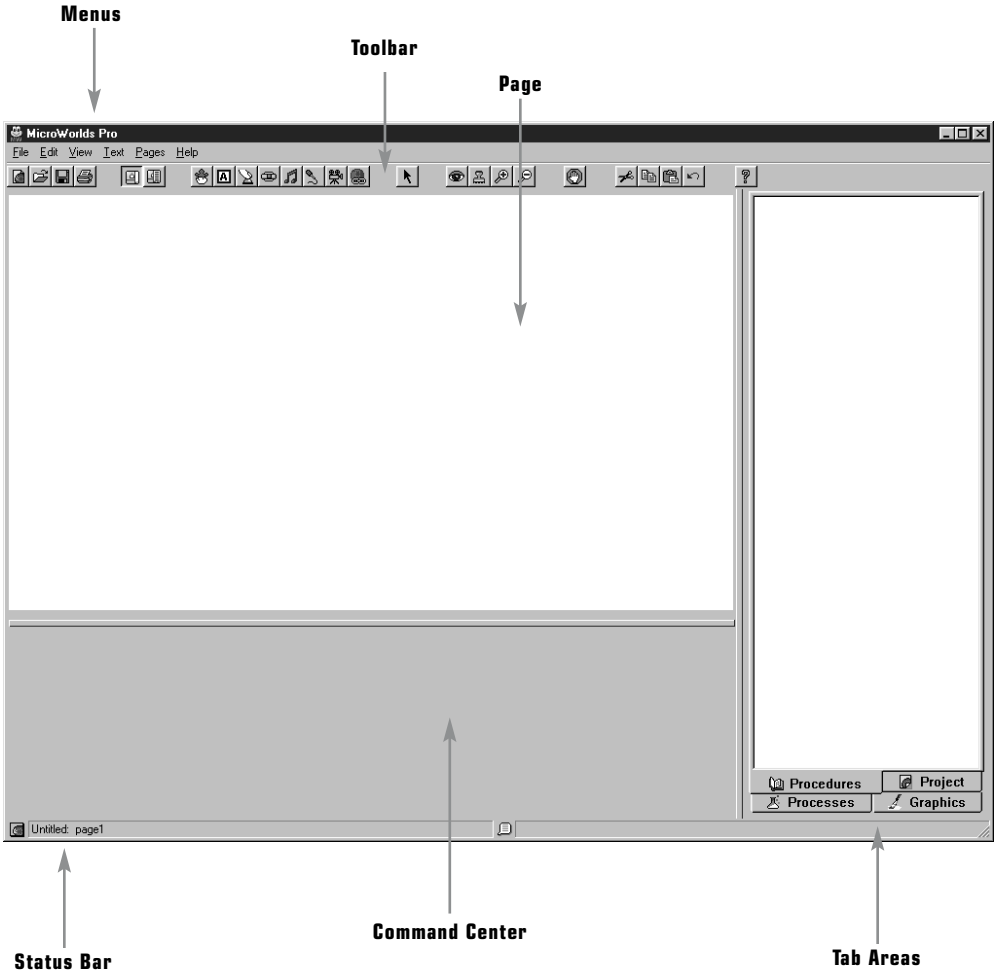
- import a picture
- insert text
- stamp text
- create special text effects
- change shapes
- animate shapes
- import music
- create a button
- create a transition

First, choose New Project from the File menu. (If you don't see the File menu, click **Esc** to get out of Presentation Mode.)

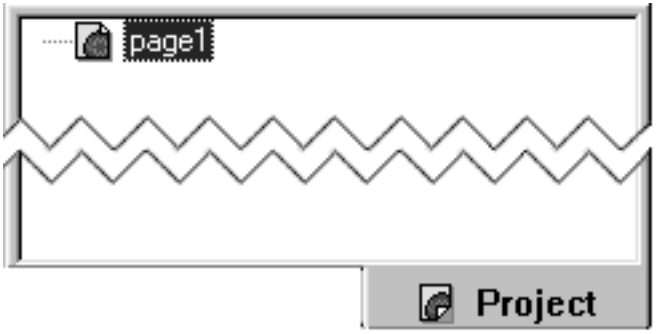
**Note**

If you still have the Welcome project on the screen, MicroWorlds Pro will ask if you want to save any changes to the current project. This will be a useful reminder for you later. At this point, however, you should click No so that the project remains unchanged.

Each MicroWorlds Pro project consists of objects and text presented on separate *pages*. A new project starts out with only one page; however, you can add more. The first page of a new project looks like this.



Note the special area to the right of the page containing tabs labeled Procedures, Project, Processes, and Graphics. This is a special functions area used for a variety of purposes. Click on the Project tab. This displays a tree diagram of the objects in the project and their relationship to each other. As you add or modify these objects, the diagram is updated.




The area beneath the page is called the *Command Center*. This is where you type instructions for MicroWorlds Pro to carry out.

Let's plan for the first page of the presentation to include a message of welcome, the course title, the location of our school, some graphics, and a bit of music.

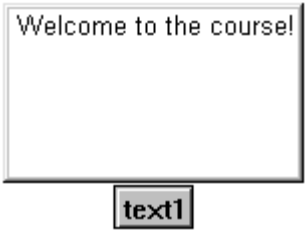
First, we'll create an interesting background for page one by importing a picture – in this case, a map that will show the location of our school. From the File menu, select Import and then Picture. In the Pictures folder, select the map file for Kentucky.

Move the map around in the window until it is where you want it.

Now create some welcoming text. Click on the Text Box tool  , and then click in the page area. Type **Welcome to the course!** in the box that appears.

**Note**

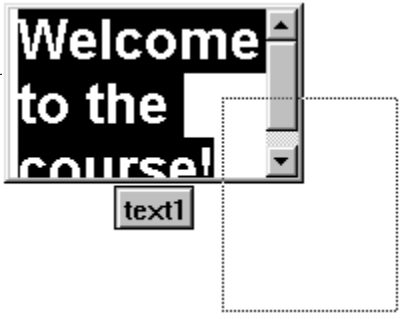
Notice that MicroWorlds Pro updates the project tree as the new text box is created and automatically names the text box Text1. This automatic naming is part of the creation of the object. Later, you learn how to name objects yourself.

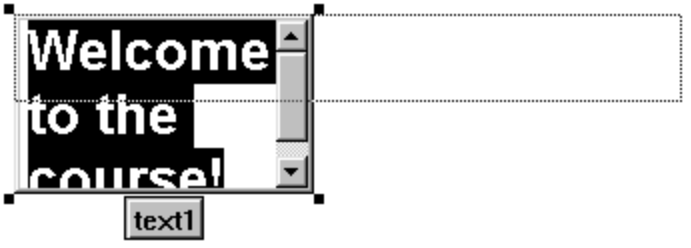



Change the style of the text to bold and the size to 20 point. To do this, first select the text and then choose Font from the Text menu, as you would with a word processor.

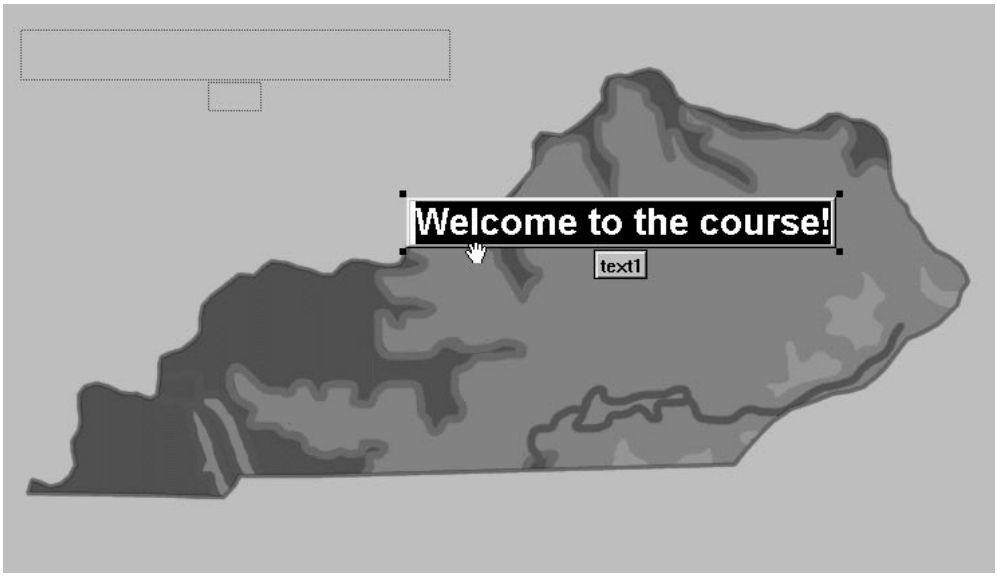

Make the text box longer so that the text will fit on one line. Here's how:

- Select the text box by dragging over a portion of it. When the text box is selected, squares called handles appear at the corners.
- Change the box size by clicking on and dragging a corner handle.



Use the text box name as a handle to drag the text box to the upper left corner.



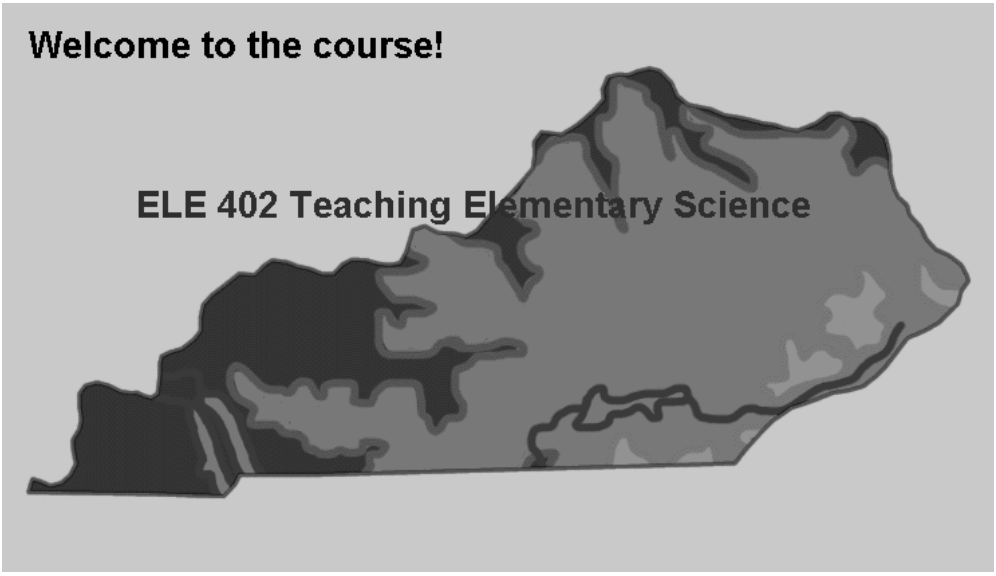

Right-click Control-click the text box and select Transparent. Now, only the text is displayed against the background. Drag the text to the desired position.

**Note**

As with many presentation programs, a number of special effects are possible for text. See More Ideas for Special Effects at the end of this chapter for details.

Now create a text box to display our course name, ELE 402 Teaching Elementary Science. Make the text bold and suitably large and then change the text to a dark color of your choice, perhaps purple.

Make the text box transparent. Drag the text to an appropriate place on the page, but make sure it does not cover up our location on the background map (Murray is located in the southwest part of the state of Kentucky).




Before going any further, save your project by selecting Save Project from the File menu. Type a suitable project name in the File Name box and click **Save**. Note that the name you give the project appears in the status bar at the bottom of the screen after you save it for the first time.

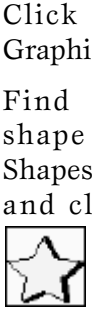
**Note**

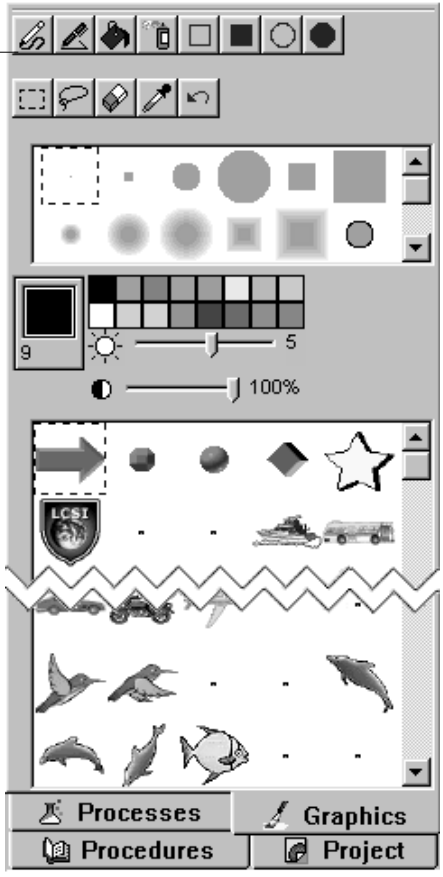
Experienced developers save their projects frequently while developing them. This saves recovery time and effort when something unexpected happens.

## GETTING FANCY

Now let's create a turtle to fancy things up a bit. The *turtle* is one of the special objects in MicroWorlds Pro. Select the Turtle tool and click on the page. Next, we'll change the shape of the turtle to a star and use it to mark our location on the map. Here's how:

- Click on the Graphics tab.
- Find the star shape in the Shapes collection and click on it.



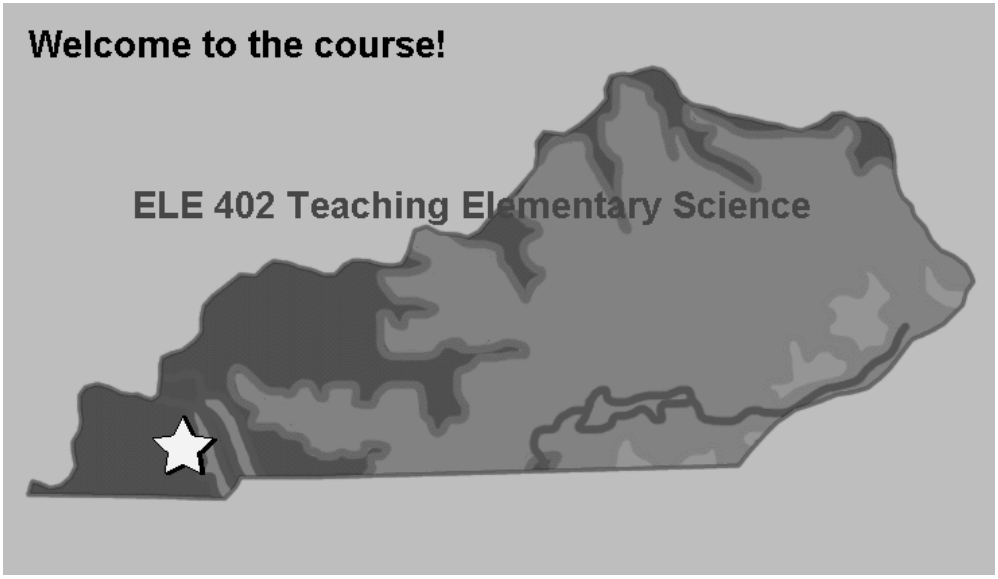



**Note**

Note that the mouse pointer changes to a hand with the index finger extended, indicating that it is ready to transfer the shape.

- Click on the turtle.

- Drag the star-shaped turtle to Murray, Kentucky, the home of Murray State University.




The turtle is a special object that carries out instructions when you click on it. Let's *program* the turtle so it blinks repeatedly when clicked. Here's how:

- Right-click | Control-click on the turtle and select Edit to open its dialog box.

**Note**

**Ht** is shorthand for **h**ide**t**urtle, and **st** means **s**how**t**urtle. One of the turtle's states is whether it is visible or not. The **ht** command makes the turtle invisible, and the **st** command makes it visible. The **wait** command makes MicroWorlds Pro pause for a number of tenths of a second. In this case, **wait 5** makes it pause for five tenths of a second.

- In the Instructions box, type the following: **ht wait 5 st wait 5**
- Click OK.

**Note**

1. **Ht**, **st**, and **wait** are examples of *commands* in Logo, the underlying computer language of MicroWorlds Pro. *Programming* a computer means using stored sequences of commands and other instructions to produce a desired action or effect. When you type **ht wait 5 st wait 5** in the turtle's Instructions box, you are storing the instructions to be carried out when the turtle is clicked. You have begun to program a computer.
2. You may wish to click on the Project Tab and observe how the project tree has changed. When the turtle has been programmed, a plus sign | right Arrow appears beside it. Click on the plus sign | right Arrow to view the turtle's instructions.

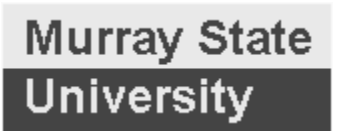
Click on the turtle in the project page. It disappears for a half-second and then reappears. Change this to continuous blinking by clicking Many Times in the turtle's dialog box. The turtle starts blinking when you click it and stops when you click it once more. This is a type of animation.

Add another text box next to the star (turtle) and type in the name "Murray State University."


Change the colors of the text to the school colors: make the words "Murray State" blue and "University" yellow. Adjust the size of the text box and then make it transparent.

Add emphasis to the colors of the text by placing contrasting-color rectangles under them. First, protect the project background by typing **freezebg** in the Command Center.

Then click on the Graphics tab. Use the paint tools to paint a yellow rectangle under the blue letters and a blue rectangle under the yellow letters.




**Note**

If you paint a rectangle and then want to remove it, use the Undo button or the Eraser tool .

## Big Finish

As a finishing touch, add music to the page:

- From the File menu, choose Import Music. Select the **Pomp** music file.
- Click on the **Pomp** icon to hear the music.


- Drag the icon to a corner of the page and then Right-click Control-click it and select Edit to display its dialog box. Uncheck Show Name and click OK, so that the Pomp name is not part of the presentation display.

**Note**

Instead of loading the music file into the project, MicroWorlds Pro makes a note of the music file location and creates an icon on the project page with a link to that music file. This is one of the many ways MicroWorlds Pro keeps the size of a project file to a minimum. It also means that you must remember to include all associated media files when you copy a project to another disk.

A page transition brings the page up on the screen in an interesting way. Click on the Pages menu, and then on Transitions. Select a transition for your page, then click OK. In a moment, you will observe the effect you have just selected.

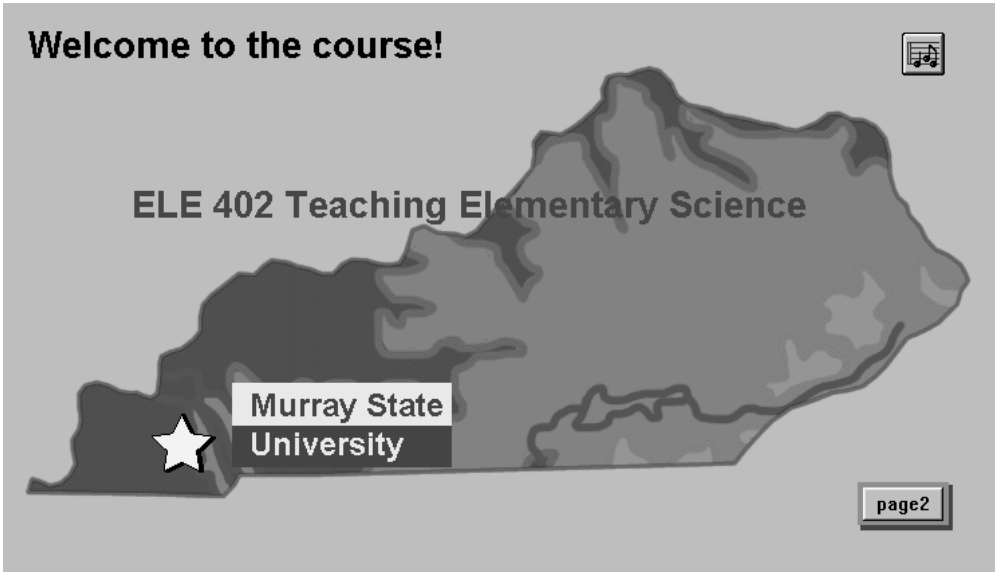
**Note**

You can change your transition at any time using the same process.

Finally, create and embellish a button to take you to the next page. Don't worry that you have not yet made a second page. You will shortly. First, do this:

- Click on the Button tool. 
- Click in the lower right portion of the page area.
- Type **page2** in the Instructions box.
- Drag the button to the lower right corner of the page.
- Paint a colored rectangle around the button for decoration.

> **Note**
>
> **Page2** is a command, so it needs to be typed as a single word with no spaces.




From the Pages menu, select New Page. This creates Page2.

Create a similar button with **page1** as its instruction and drag it to the lower left corner. 

Click on this button to go back to Page1.

> **Note**
>
> If you see an error message displayed in the Command Center when you click on a button, check in the Project Tab area to make sure that the instruction (**page1** or **page2**) is correct.

Did you like the transition? If not, select another one. Then return to Page2 using the **page2** button 

you made earlier.

## Turning the Page

We want to use the second page to explain the course goals. Let's include some blinking text, a bit of hidden text, and a colorful border as well.

In setting up Page2, you will learn how to:

- change the color of the background
- stamp repeating patterns
- make text appear and disappear
- write simple procedures (small computer programs)
- launch processes to play music
- use turtles to display and hide text

First, let's set the background to a color. How about a nice shade of blue? In the Command Center, type the following:

**setbg "blue**

> **Note**
>
> The **setbg** command **set**s the **b**ack**g**round to a specified color. Note the use of the opening quotation mark that precedes the color name and is not followed by a closing quotation mark. This syntax indicates to Logo that "blue is data, not a command word.

Try a few other color names if you like.

**Note**

1. You can also use color numbers. Try **setbg 95** for example. Note that no quotation mark is used with a number.
2. In the Graphics Tab area, the default intensity setting of each color is a medium value. The number 5 on the slider beneath the color palette is at the midpoint of an intensity scale running from 0 to 9. The number for our current blue background is 105. To try a less intense version of blue, type **setbg 103**. Change the slider setting to 3 to see this in the palette. For a more intense version, type **setbg 108**.
3. Beneath the Intensity slider is a slider that adjusts the transparency value for each color. You might want to experiment with this as well. However, the transparency property is best seen when you are painting areas of different colors that overlap or overlay each other.

After experimenting, return to **setbg 105** or **setbg "blue**.

**Note**

Only the hues ending in 5 have color names that can be used with **setbg**.

## DECORATION

Now let's add some decoration to the page. First, create a new turtle  to learn how to move it; then you can create a decorative pattern by stamping a shape.

Right-click Control-click the turtle and select Edit to display its dialog box. Name the turtle **deco** and click OK.

> **Note**
>
> The name of a turtle followed by a comma (**deco,** ) tells MicroWorlds Pro that the instructions which follow are for that turtle.

In the Command Center, type the following and watch the turtle:

**deco, forward 50**

The turtle moves up the screen a short distance. 

> **Note**
>
> **Forward** and its abbreviation, **fd**, are commands that tell the turtle to move a certain number of "steps" (50 in these examples) in the direction it is heading.

Now type

**fd 50**

If you want the turtle to move in another direction, you must first change its heading. Type the following in the Command Center and watch the turtle:

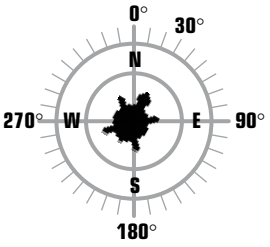
**seth 90**

Then type

**fd 50**

**Note**

The **seth** command **set**s the **h**eading of the turtle to a given azimuth, corresponding to the directions on a compass. In the example above, **seth 90** made the turtle face east (toward the right side of the page), and **fd 50** made the turtle move in that direction.




Now let's work on stamping a shape. In the Command Center, type

**Note**

The **stamp**ed turtle shape becomes part of the page graphics. Using the Eraser from the Paint tools  , you will be able to erase the stamped turtle shape, but not the actual turtle. Be sure to click the Arrow tool after you are finished with the Eraser.

**stamp**

With the mouse, drag the turtle down the page a bit to uncover the stamped shape.

Now stamp and move the turtle repeatedly. Type

**repeat 3 [stamp fd 50]**

**Note**

The **repeat** command runs the instructions in the square brackets for a specified number of times. In this example, the instructions tell the turtle to **stamp** its shape and then move **forward 50** steps. Make sure you use square brackets [ ] and not parentheses ( ) or curly brackets { }.

As long as the turtle is heading east, you can use the **repeat** command to create a line of stamped shapes going from left to right.

Let's clean up the page before we start. Use the Eraser tool  to erase all the stamped shapes. Then click on the Arrow tool and drag the turtle to a position near the upper left corner of the page.

Change the turtle's shape to the shield.


Experiment with the following line in the Command Center until you can make the turtle stamp a line of shapes across the top of the page:

**repeat 8 [stamp fd 50]**

Drag the shield turtle back to the left side of the page. Change **8** to a higher number and try again. Keep changing the number of repetitions until you have an instruction line that makes the turtle **stamp** shapes all the way across the top of the page. Use the Eraser to erase any unwanted results.

Drag the turtle back to the lower left corner and run the line of instructions to stamp the shape across the page. This is looking good!


Hide the turtle by typing the following in the Command Center:

**deco, ht**

Finally, use the Paint tools to create a brightly-colored upper and lower band for the page.


Now that you have the page decorated, freeze the background again using **freezebg**.

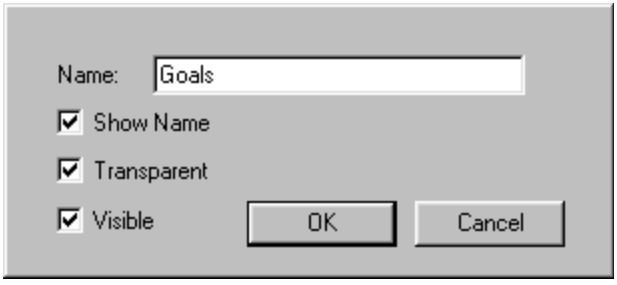
Just a reminder: have you saved the project recently? Don't forget to save regularly.

## Time for Text

Now let's add the text about course goals and activities. We want these items to blink when we click on a bullet-turtle next to them.

Add a text box and type **Course Goals** into it. Make the text bold and of a size that is easy to read (20 point or larger works well). Resize the box so that the text is all on one line.

Move the text box near the top center of the page. Right-click Control-click on the text box and select Edit to open its dialog box. Name the text box Goals and make it transparent.




**Note**

You have already seen that naming a turtle makes it easy to give it instructions. This same idea works with text boxes also, as you will see shortly.

Create a second text box with **Course Activities** in it. Name the text box Activities.


Make the text transparent.

Create a new turtle by clicking on the Turtle tool and then in the page. Drag the turtle to the left of the Activities text and change it to one of the bullet shapes. Name the turtle if you wish.


How do you get the Activities text to blink when you click on the bullet-shaped turtle? One way is to write a procedure (a computer program; usually a short one). Here's how:

Click on the Procedures Tab and type the following in the blank area:

```
to blink
hidetext
wait 5
showtext
wait 5
end
```

**Note**

1. All procedures start with the word **to** followed by the procedure name of your choice on the same line. Each procedure must have a one-word name; in this case, **blink**. Next come the instructions. Finally, the word **end** on a line by itself signals the end of the procedure.
2. In the blink procedure, **hidetext** and **showtext** are commands to hide and to display the text.

Open the turtle's dialog box and click Many Times. Then type in the Instructions box:

**Activities, blink**

**Note**

The name of a text box followed by a comma (**Activities,** ) tells MicroWorlds Pro that the instructions which follow are for that text box. The name of the **blink** procedure is now an instruction which is carried out by MicroWorlds Pro; it runs all of the instructions in the **blink** procedure.

Click on the turtle to see the Activities text **blink**. Click on the turtle once more to stop the blinking.

**Note**

It is possible that the Activities text might be invisible when you stop the **blink** procedure. This is because you clicked on the turtle while the text was hidden. To make it visible, click on the turtle and then click again when the text is displayed.

With the Goals text box, let's do something a little fancier. We'll play a melody while blinking the text a certain number of times.

First, create another turtle and drag it to a position next to the Goals text. Change it to a shape of your choice.


Then create a melody of your own. Here's how:

- Click on the Melody tool and then in the page to display the melody keyboard.
- Play a brief tune on the keyboard with the mouse, then click on Play to listen to it. Select different instruments until you find one you like. When you have finished with your tune, replace the default name of melody1 with Doremi and click OK.
- Click on the resulting Doremi icon that appears on Page2 to play your melody once more.
- Drag the icon to the side of the page.

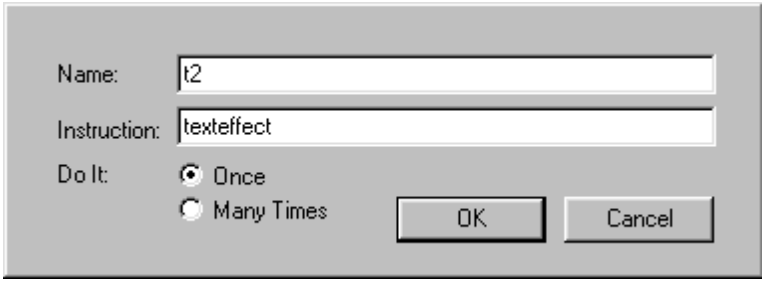
Now let's write a procedure to play your melody and make the Goals text blink. Type this in the Procedures Tab area:

```
to texteffect
doremi
goals, repeat 5 [blink]
end
```

**Note**

The second instruction directs the Goals text box to **repeat** the **blink** procedure **5** times. A procedure such as **texteffect** is sometimes called a *superprocedure* because it runs other procedures.

Return to Page2 and open the dialog box of the turtle next to Goals. Type **texteffect** in the Instructions box and click OK.




Then click on the turtle to run the **texteffect** procedure.

Note that the text starts blinking only once the music has stopped. If we could get the text to blink while the music was playing, the effect would be more striking. To do this, revise the **texteffect** procedure as follows:

```
to texteffect
launch [doremi]                    ← this line changed
goals, repeat 5 [blink]
end
```

When everything is working the way you want, hide the melody icon. Right-click Control-click on the melody icon in the Project Tab area and select Hide.

**Note**

When running a procedure, MicroWorlds Pro carries out the instructions one at a time. However, MicroWorlds Pro is capable of what is called *parallel processing*. This means that it can launch a procedure or any other action (such as playing a melody) as an independent process to be run on its own, and at the same time carry out other tasks. The **launch** command runs its input (in square brackets) as an *independent process*. In the changed **texteffect** procedure above, MicroWorlds Pro launches the Doremi melody and then proceeds immediately to the next instruction, making the Goals text **blink**.

## Going Undercover

As the final item on the page, let's create an example of how to "uncover" and "cover up" a line of text. We'll do this with information about one of the course goals. For this, we will need a transparent text box with text the same color as the background. We will also need two turtles, one to uncover the text and the other to cover it back up.

First, create a new text box and type the following in it:

**Increased interest in science.**

Name the text box Interest. Change its font to be slightly smaller than the others so the text can be easily "covered."

While the text is still selected, change the text color to the same shade of blue as the background by typing the following in the Command Center:

**interest, settc "blue**

> **Note**
>
> **Settc set**s the **t**ext **c**olor to the input which can be either a color name or number. In this case, you must use the **settc** command to match the text color with the background color. If you use the color item in the Text menu, the text color will be different.

Create a new turtle and name it Uncover. Drag it to a position to the left of the Interest text. Change its shape to an arrow.

Now, make the text box transparent so that the text disappears into the background.

Now let's modify the Uncover turtle so that its heading is to the right and it can draw a wide yellow line. Type the following in the Command Center:

**uncover, seth 90**
**setc "yellow**
**setpensize 30**
**pd**

**Note**

You have just operated on four different *turtle states*. **Seth 90** changed the *heading* of the turtle so it faces to the right (east). **Setc "yellow** sets the *color* of what the turtle draws. **Setpensize 30** sets the *width of the line* the turtle draws. **Pd** sets the state of the *pen* to down; that is, it enables the turtle to draw a line as it moves.

Still in the Command Center, type

**glide 400 1**

**Note**

The **glide** command moves the turtle a total distance (the first number input) at the speed indicated (the second number input).

The Uncover turtle moves slowly across the screen from left to right, drawing a wide yellow line. If you positioned the turtle correctly to the left of the hidden Interest text, you will be able to see the "uncovered" text highlighted by the yellow background of the line the turtle drew.

Type in the Command Center:

**glide -400 1**

**Note**

There is no space between the negative sign and the number here, because we want to pass a negative number to **glide** as input. This makes the turtle move in the opposite direction without having to change its heading.

The Uncover turtle returns to its original position. Let's incorporate these two moves into a procedure. Type the following in the Procedures Tab area:

```
to paint
glide 400 1
glide -400 1
end
```

Then type **paint** in the Instructions box of the Uncover turtle. Click on the turtle and observe its movement. Adjust its position by dragging it so that it "uncovers" the Interest text in just the way that you want.

To remove the yellow paint for the moment, click on the Graphics Tab. Select the Paint Can tool  and the same blue color as the background. Click in the yellow area to fill it with blue.

## Cover Up

Now let's develop another turtle to cover the text back up by drawing a wide blue line over the yellow paint.

Create a second turtle and name it Cover. Type **paint** in its Instructions box.

Now let's get it positioned accurately to cover the Interest text. Type the following in the Command Center:

**cover, setpos uncover's "pos**

**Note**

**Setpos** is a command that sets the position of a turtle. **Uncover's "pos** is a *reporter* that outputs the exact position of the Uncover turtle. Note the single set of quotation marks used with **"pos**. This tells MicroWorlds Pro the name of the state (i.e. the **pos**ition) that is required. The result is that the Cover turtle is placed at exactly the same location as the Uncover turtle. For more information, see **setpos** and **pos** in the Help Vocabulary.

Now set the other states of the Cover turtle by typing the following in the Command Center:

**cover, seth 270**
**setc "blue**
**setpensize 30**
**pd**

> **Note**
>
> **Seth 270** sets the heading of the Cover turtle so it points toward the left side of the page (west).

The Cover turtle should have the same shape as the Uncover turtle, but it should be a mirror image. Here's how to prepare such a shape.

- Click on the arrow shape in the Shapes palette to select it. 
- Copy and paste the shape onto an empty shape. 
- Right-click Control-click the pasted shape and select Edit to display the Shape Editor.
- Click on the Mirror Image icon  to flip the shape left over right. 
- Click OK to exit the Shape Editor.

Now change the shape of the Cover turtle to that of the new mirror image arrow. Then type the following in the Command Center to place the Cover turtle at its starting position.

**cover, back 400**

> **Note**
>
> **Back 400** moves the Cover turtle **back**wards to the end of the Uncover turtle's line of travel.

**Note**

You could use a copy and paste technique for additional text. Drag over the two turtles and the Interest text to select them, then copy and paste them as a group onto the page. With all three objects still selected, drag the group to the desired location (for example, beneath the Activities text). You must make the text opaque before you can edit it.

Click on the Uncover turtle to display the Interest text. Click on the Cover turtle to hide it again.

Click on the Pages menu and select a transition for your page.

As a final touch, create a button with **page3** as its instruction and drag it to the right side of the page. Decorate it as you wish.


Create Page3 by selecting New Page from the Pages menu. Create a **page2** button and use it to return to Page2 so you can see your transition. page2 Then click the **page3** button. page3

There you have it! On Page2, you have changed the color of the background, stamped repeating patterns, made text appear and disappear in several different ways and used bullets to accent and activate text. You have also written procedures. With these skills, you can design many different effects for your own pages.

## Animated Text

For Page3, you can work on some interesting text movement, and then finish up as you wish.

This page contains information about you, one of the course teachers. In one area, your name, telephone number, and e-mail address move back and forth. In another area, a message to students appears and scrolls. In the background, energetic music plays.

Let's import a presentation background before setting up the text animation. Choose Import Picture from the File menu and select Presen1 in the Pictures folder.

To make text move around the screen, the best technique is to copy the text and paste it onto a turtle as a shape. For example, here's how to set up your name:

- Create a text box and type in your name.
- Make the text bold and at least 16 point in size.


- Make the text box transparent.
- Click on the Stamp tool  and click on the text box to stamp the text onto the page.
- Drag the text box away from the stamped text.


- Drag a selection rectangle around the stamped text using the rectangle Selection tool in the Graphics Tab area.


- Copy the stamped text and paste it onto an empty shape. 


- Create a new turtle and name it Profname. Set its shape to that of the text.

Use this same sequence to create a text-shaped turtle for your telephone number and e-mail address.

Erase all stamped text by double-clicking the Eraser Tool. In the Project Tab area, Right-click Control-click on each of the text box objects in the page and select Remove.

Drag the text-shaped turtles into the central area of the background. Now, draw a wide-bordered red (color 15) frame around them. This will form the boundary of their movement area.

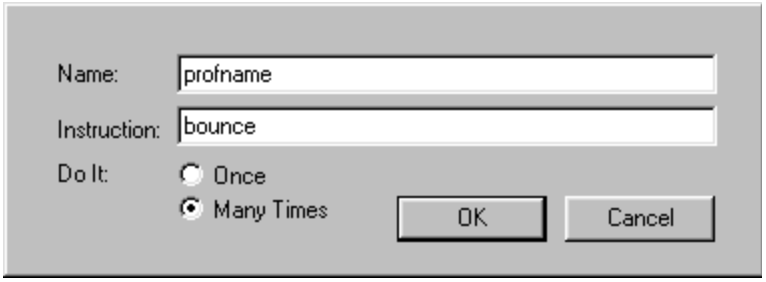

Write a procedure that makes the text-shaped turtles "bounce" when they encounter the red wall.

```
to bounce
if colorunder = 15 [right 160 + random 20]
fd 1
end
```

**Note**

**Colorunder** = **15** reports **"true** to **if** when a turtle encounters the red wall of the rectangle.
**Right 160 + random 20** changes the direction of the turtle's travel, but also includes a random component as well, just to make things interesting.

In the dialog box of each text-shaped turtle, type **bounce** in the Instructions box and select Many Times. Click OK and then click on the turtles to observe the movement!




**Note**

The active area of a text-shaped turtle is at the center. If you click near the edges, the turtle will probably not respond.

Now let's set up some scrolling text. Name a new text box Notes, and type in some information for our students. Here is an example.

**Anytime you have a question about the course, please contact me. I check my e-mail several times each day. I try to return telephone calls promptly. I want to help you accomplish the professional goals you set for yourself.**

Make the text bold and at least 16 point size. You may also wish to change it to a favorite color.

Resize the text box so that each text line has about three to four words on it. Put a carriage return between each sentence for spacing. Drag the box to a position at the bottom of the page.

In the dialog box, uncheck Show Name and Visible.

Now write a procedure to make the text scroll:

```
to scroll
notes, top
showtext
repeat 12 [cd wait 15]
hidetext
end
```

**Note**

**Notes, top** places a text cursor at the top of the **notes** text box. **Showtext** makes the box visible. **Cd** moves the **c**ursor **d**own one line. **Hidetext** makes the text box invisible once more. You may have to adjust the **repeat** number for the number of lines in the text box.

To observe the effect, type **scroll** in the Command Center. Don't worry that the text disappears after the procedure runs. It will come back each time you **scroll**.

For fun, let's bring in some energetic music. From the File menu, select Import Music. In the Media folder, select Overture. Click on the resulting icon to hear the music.

Now let's write a superprocedure to start everything:

**Note**

First the music is **launch**ed. The text in the red square springs into action after a **wait** of 15 seconds. **Everyone [clickon]** makes all turtles act as if they have been clicked. The **notes** text appears and scrolls, then disappears. **Everyone [clickoff]** shuts off the text-shaped turtles.

```
to go
launch [overture]
wait 150
everyone [clickon]
wait 50
scroll
everyone [clickoff]
end
```

Create a **go** button for Page3, click on it, and watch the show!


Make any desired changes to the **wait** times in the procedures. Click on the Processes Tab to see which processes are running (for more about processes, see Process Management in Help Topics).

Embellish the page in any fashion you wish, using paint tools, imported graphics, or other techniques you have learned. Select a transition for the page. (And don't forget to save the project!)

On this page, you have created moving turtles, used a color to contain them, created scrolling text, and written a superprocedure to control the starting and stopping of the whole thing. Not bad!

## More Ideas for Special Effects

If you have used presentation software before, you are aware of drop shadow special effects. Here is a brief checklist of one way to create similar effects in a MicroWorlds Pro project:

For drop shadow text:

- Make the text box transparent.
- Using the Stamp tool, stamp the text onto the page as a graphic.
- Drag the text box away slightly.
- Change the color of the text in the text box and then stamp it onto the page.


For drop shadow background:

- Protect the existing background with **freezebg**.
- Make the text box transparent.
- Paint a filled black rectangle over the text.
- Paint a filled rectangle of another color slightly above and to the left of the black one.
- Drag the text into the colored rectangle.

## SUMMARY

In this chapter, you have explored a sample MicroWorlds Pro project and then created a few pages of a presentation project. In doing this, I hope you have learned a few of the features and tools of MicroWorlds Pro. You are now ready to add more pages and information to this project, or to create your own.

Did you know that animation is possible with MicroWorlds Pro? Chapterette A contains some useful techniques and extension ideas.

Chapter 2 will introduce you to the interactive side of MicroWorlds Pro; its chapterette will show you how to get your projects onto the World Wide Web!

# *The Animation Station*

**WELCOME TO ANIMATION! In this chapterette, you will explore several different ways to create animation to liven up your MicroWorlds Pro projects. Most animation is done with the turtle: by getting it to move, change shapes, or both.**

## MOVEMENT

Let's suppose that you want a bus shape to move across one of your project pages. Go to a new page and create a turtle. Change its shape to a bus.

Now, Right-click | Control-click on the turtle, and choose Animate.

**Note**

To give the turtle a bus shape, click on the Graphics tab, click on the bus shape, and then click on the turtle. Alternatively, you could type **setshape "bus** in the Command Center. The **setshape** command **set**s the turtle to a **shape** you designate. For more details on **setshape**, see Vocabulary in the Help menu.

The turtle moves continuously forward towards the top of the page. To change its direction, press the Shift key | ⌘ key and drag it towards the right.

**Note**

To change the movement direction of the bus-shaped turtle, you can use the **seth** command to **set** the **h**eading of the turtle to the direction you wish. The default setting is 0, or north, toward the top of the page. East is 90, south is 180, and west is 270. To have the turtle move in a westerly direction, type **seth 270** in the Command Center.

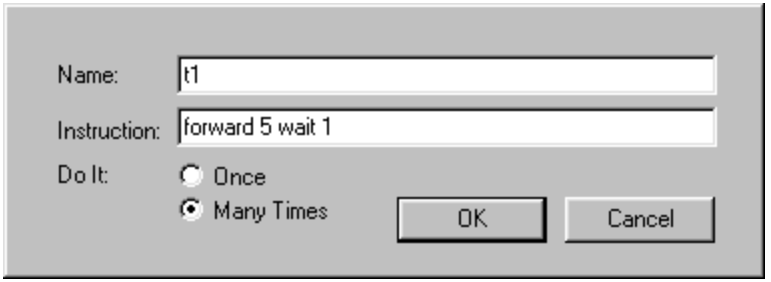
If the bus is moving too fast or too slow, stop it first.

**Note**

To stop your continuously moving turtle, you can do one of several things. You might want to try each of these methods, one at a time.

- Click on the turtle (if you can!).
- Click on the Stop button in the Toolbar.
- Press Ctrl-Break | ⌘ .
- Type **stopall** in the Command Center.
- Right-click | Control-click on the turtle's instruction in the Processes Tab area and select Stop.

Then Right-click Control-click on the turtle and select Edit to open its dialog box. You'll see the instruction that Animate automatically inserted.




Experiment with changing either the **forward** input or the **wait** input. Click OK.

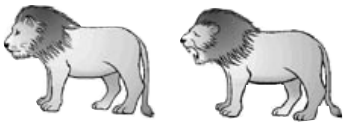
Create other turtles, give them interesting shapes, and set them in motion.

**Note**

You can also control your turtles from the Command Center. For example, type **everyone [clickon]** and watch all turtles that have movement instructions act as if they were clicked. Use **everyone [clickoff]** to stop all the turtles.

## MOTION

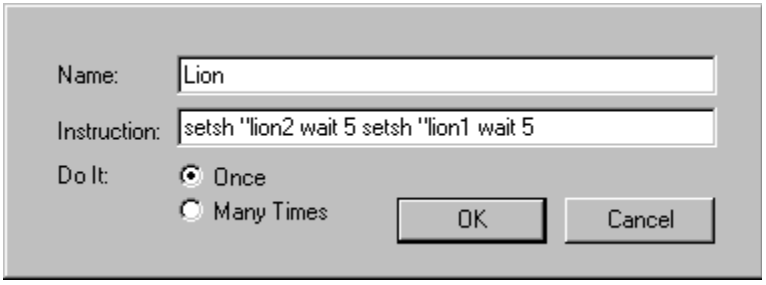
The motion of the mouth of an animal opening and closing illustrates another type of animation. In a MicroWorlds Pro project, you can create the illusion of this motion by changing shapes. Let's program a turtle to look like a lion opening and closing its mouth.



In the Graphics Tab area, locate the two lion shapes. The name of each shape appears when you leave the mouse pointer on it. On Mac, the Help on shape names can be obtained using the Help Balloon. Note that for both Mac and IBM, simply clicking on a shape will show its name in the status bar at the bottom of the project.

Create a turtle and name it Lion. In its Instructions box, type the following, then click OK:

**setsh "lion2 wait 5 setsh "lion1 wait 5**




**Note**

1. Because this is a long line of instructions, you may wish to include them in a procedure and then type just the procedure name into the Instructions box.
2. **Setsh** is an abbreviated form of the **setshape** command. Experiment with the number used with the **wait** command. **Wait 1** creates a faster sequence, while **wait 10** makes it look like the lion is yawning!

Each time you click on the turtle, the lion changes shape and appears to be roaring. This is an example of programming the turtle for animated motion.

**Note**

You may wish to add a roaring sound by importing a sound file or using the microphone. Include the **launch** command in the Instructions box to start playing the sound.

## All Together Now

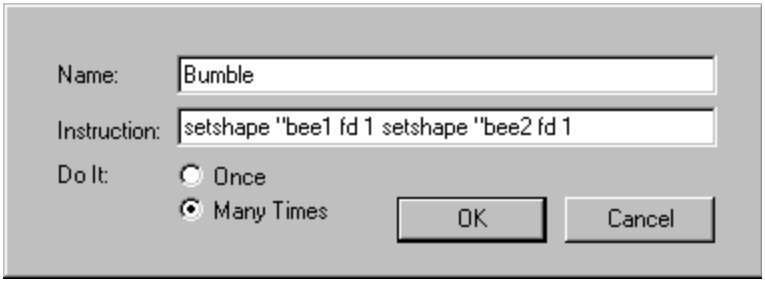
When you combine the movement of the object and the motion of its parts, then you have yet another type of animation. As an example, let's make a bumblebee flap its wings as it moves along. Here's how you can do this:

Create a turtle and name it Bumble. In the Instructions box, type the following:

**setshape "bee1 fd 1 setshape "bee2 fd 1**

Then select Many Times and click OK.

> **Note**
>
> Recall that **fd** is the abbreviated form of the **forward** command.




Click once on the turtle and watch it carry out the instructions you typed, continuously changing shapes and moving forward.

This is an example of programming the turtle for full animation, with both movement and motion. You can use another series of shape names instead of **bee1** and **bee2** to create similar effects.

> **Note**
>
> Use the **seth** command in the Command Center to **set** the **h**eading of the turtle to the direction in which you want it to move.

## Automated Animation

Here is another way to program full animation. Let MicroWorlds Pro remember the sequence of shapes instead of you having to type them in.

Go to the Graphics Tab area and find the series of four galloping horse shapes. Their names are **horse1**, **horse2**, **horse3** and **horse4**.

Create a turtle on the page. Turn the turtle so it is facing a horizontal direction (heading 90 or 270). Hold down the Shift key ⌘ key while you click on the horse shapes and the turtle in the following sequence: *horse1, turtle, horse2, turtle, horse3, turtle, horse4, turtle*.

**Note**

By holding down the Shift key ⌘ key as you change the turtle shapes, you are telling MicroWorlds Pro to associate the *sequence* of horse shapes with the turtle. Each time the turtle encounters a movement command (**forward**, **back**, or **glide**), it will change to the next shape in the series.

Type **fd 10 wait 1** in the turtle's Instructions box and select Many Times. What do you think will happen when you click on the turtle?

What do you think will happen if you type **glide 200 10** in the Command Center?

## Making a List

You can also use the **setshape** command to give the turtle a sequence of animation shapes. For example, create a turtle and name it Buckskin. Then type the following in the Command Center:

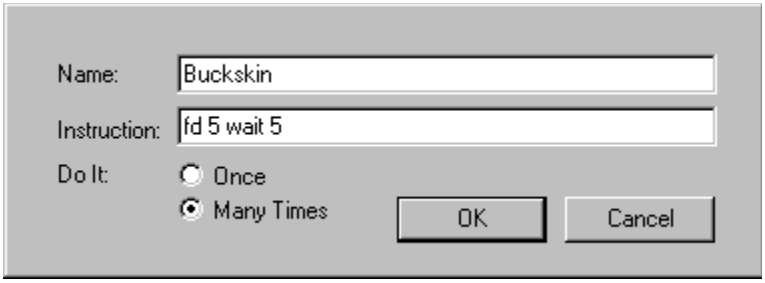
**buckskin, setshape [horse1 horse2 horse3 horse4]**

**Note**

This is an example of programming a turtle using the **setshape** command with a list of shapes enclosed by square brackets.

Any time the Buckskin turtle encounters a movement command, it will change to the next shape in the list. Here are a couple of things to try:

- Type **buckskin, glide 100 10** in the Command Center.
- In the Instructions box of the turtle type **fd 5 wait 5**, and select Many Times.




**Note**

To cancel a list or series of shapes for a turtle, change the shape of the turtle to a single other shape.

What other ideas can you think of?

## Animation Your Way

By now, you are probably wondering if you can create animation with shapes you create yourself. Of course! To create a new shape, double click on one of the empty shapes in the Shapes palette, and then use the paint tools to design the shape.

**Note**

You can also use Copy and Paste to transfer graphics and stamped text to turtle shapes.

Once you have a series of shapes, use any of the above techniques to animate your creations!

## Flipping Through

Have you ever seen a "flip book" of cartoon drawings? This is usually a small book with 30 to 60 pages of a sequence of cartoon character drawings. As you flip the pages, the character seems to move. A similar frame-by-

frame technique has been used by film and video cartoonists for more than 50 years.

We can adapt this technique for use with MicroWorlds Pro. First, design a series of at least five consecutive frames showing simple animation of some sort. Perhaps you want to use a stick figure, for example, or a car moving across the page.

First of all, start a new project. Then use the paint tools to draw the first frame of your animation project.

When you have completed your drawing, select Duplicate Page from the Pages menu. This creates an identical copy of the page you are working on. Change the graphics of the new page to that of your second animation frame.

Continue this process until you have the consecutive frames you have planned for.

**Note**

Usually, you will make only small graphic changes from one frame to the next. Since you are starting your subsequent frame drawings with a duplicate of the previous page, you will not have to redraw very much. Simply change the objects or parts of objects that are animated.

In the Procedures Tab area, type the following procedure:

```
to flip
dolist [ i pagelist ] [ getpage :i wait 5 ]
end
```

**Note**

The **flip** procedure is brief, but it contains some programming concepts that may require explanation. The **dolist** command is one that may be unfamiliar to you. Its purpose is to run a list of instructions using a range of variable values. Here, it creates a variable **i** and cycles through the **pagelist**, assigning each page name to **i** in turn. As **:i** assumes each value, **dolist** runs the list of instructions **[getpage :i wait 5]**. This displays each page in sequence. For more details on the **dolist** command, see the Vocabulary section in the Help Menu.

**Note**

1. To see the order of pages in the **pagelist** reporter, type **show pagelist** in the Command Center.
2. You might wish to adjust the length of the **wait** interval in the **flip** procedure.

The **flip** procedure can be run from the Command Center. Display the first page in your sequence, and then type **flip** in the Command Center. Enjoy!

## In Conclusion

You have explored the three types of animation with shapes: movement across the page, motion of a part, and movement and motion combined. You have also learned how to program the turtle for a sequence of shapes by holding down the Shift key ⌘ key when changing shapes, and by using a list of shapes with the **setshape** command. Finally, you have experimented with the idea of "flipping" pages to produce animation.

As you gain more experience with these techniques, you will also get a sense of which technique is the best to use for a particular situation.

## Extensions

Apply the concepts and commands you have just learned to explore these additional animation ideas.

- Animate a shape that changes size.
- Animate text or letters of the alphabet so that they move across the page.

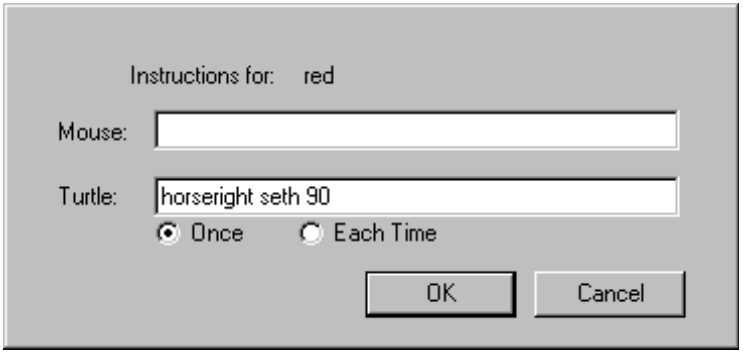
- Animate shapes that change colors gradually.
- Animate a rainbow that cycles through several patterns of colors.

Here is a mini-project you might enjoy: a horse that gallops back and forth between two colored bars on the page. Do this to get started:

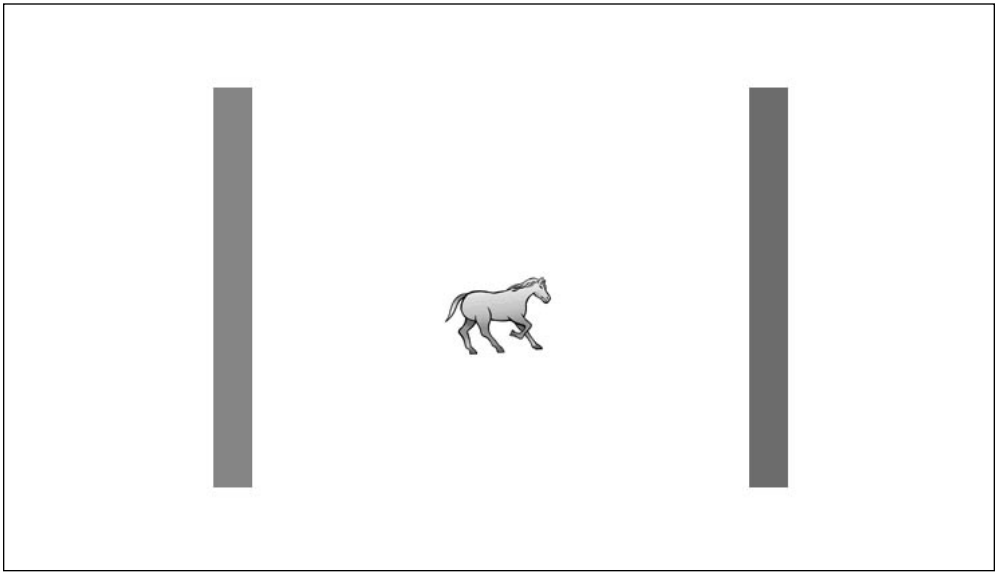
Copy the horse shapes one at a time, paste them to empty shapes, and name them **horse5**, **horse6**, **horse7**, and **horse8**. Flip them so that this set of horses faces left. Define the following procedures:

```
to horseright
setsh [horse1 horse2 horse3 horse4]
end

to horseleft
setsh [horse5 horse6 horse7 horse8]
end
```

Right-click | Control-click on the red color in the Graphics Tab area. In the dialog box, type **horseright seth 90** for the *Turtle* instruction.



Then draw a wide vertical red line down the left side of the page. Right-click | Control-click the violet color to open its dialog box. Type **horseleft seth 270** for the Turtle instruction, and draw a vertical violet line down the right side of the page.



Create a turtle between the lines. Set its heading to the right. In the Command Center, type **horseright**. Then Right-click | Control-click on the turtle, choose Animate, and watch!

CHAPTER 2

# *Interacting …*

## Overview and Objectives

**IN THE FIRST CHAPTER, you began to learn about MicroWorlds Pro by developing pages for a presentation. In this chapter you will create an interactive project that will help you learn more about programming and project development. Along the way, you will learn how to create objects and write procedures that:**

- **Tell turtles to play notes and change shapes when clicked.**
- **Pick notes randomly and keep track of them.**
- **Keep track of the notes *you* play.**
- **Compare your notes with those picked by the computer.**
- **Report the results.**
- **Keep score of your performance.**

The interactive project in this chapter uses repetition as a way to assist memory. Specifically, the project is a version of a popular electronic musical game called SIMON®. On the screen are four objects of different colors. The player must click on the objects in the order selected by the computer. Each time a colored object is clicked, it blinks and plays a musical note.

To start, the computer picks one of the colored objects at random and makes it blink and play a note. The player clicks on the same object. The computer repeats the first object and follows with a second object and note, selected at random. The player must then click on the two objects, in the right order. After verifying that this was done correctly, the computer adds a third object to the sequence, and then a fourth, and so on. The player's goal is to click on the objects in the correct order for the longest possible sequence. When one of the objects in the sequence is incorrect, the game is over. Although all human players are destined to fail at some point, the length of the sequence some players can remember is surprising. This is because of the repetition of the sequence in each round of play.

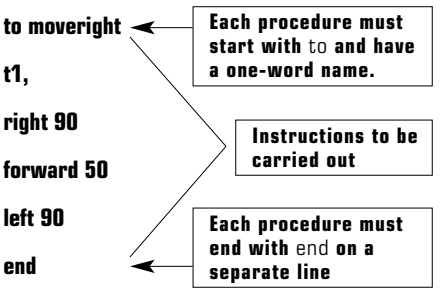
Developing an interactive project like this one will help you learn more about the power and potential of computer programming. Through fairly simple programming, you will be able to initiate and manage complex actions, and have some creative control at the same time.

The project development process is also important to learn. Although the project might appear somewhat complicated to you initially, it can be developed in a series of smaller steps. Procedures (small programs or sequences of instructions for MicroWorlds Pro) are the building blocks that make this possible. As you progress through Round 1, Round 2, etc., notice how each section builds on the work of the preceding one.

**Note**

In this chapter, you will be editing or changing procedures frequently. Here is a quick review of procedures:

- Each procedure has a one-word name. This name is interpreted as a command, instructing the computer to carry out the procedure instructions. The procedure name can be typed in the Command Center, in the instruction line of dialog boxes (e.g., for turtles, colors, or buttons), or as commands within other procedures.
- To write a procedure, type in the Procedures Tab area. Here is a sample procedure:




- When you are finished with a procedure, you can run it by typing its name in the Command Center or other appropriate location. If your procedure does not work as you wish, or you see an error message (e.g., I don't know how to ___ ), then examine it and make any appropriate changes. After you activate the Procedures Tab area, pressing F4 will tell MicroWorlds Pro to run a grammar check on the procedures, highlighting some common errors.
- For additional details, select Logo Programming from the MicroWorlds Pro Help Topics in the Help menu.

Let's get started!

## Round 1: Worthy of Note

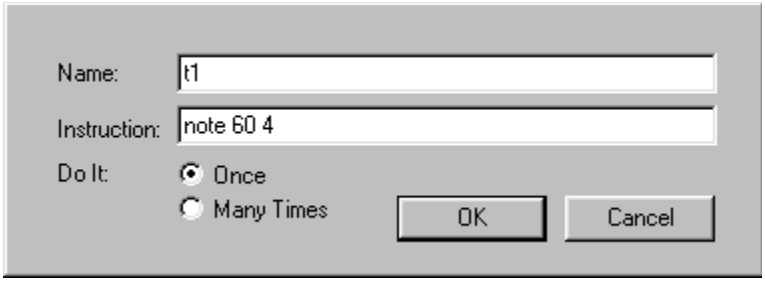
Our first task is to create turtles that play notes and change shapes when clicked.

Start a new project.

If you are planning to work through Chapterette B and upload your project to the World Wide Web, then select New Project Size and Web Player now. This will keep your project at a reasonable size for Web publishing. It will not affect your work in this chapter in any obvious way.

First, let's program the turtle to play a note when you click on it. Create a turtle, type the following instruction into its dialog box, and then click OK. Be sure to put spaces between the two numbers as shown:

**note 60 4**




Click on the turtle to hear the musical note.

**Note**

1. If you do not hear a musical sound when you click on the turtle, make sure your speakers are turned on and that your system is functioning properly.
2. The first input number of the **note** command is the MIDI[1] note number. **60** is the MIDI number for middle C. The second input number is the note duration in tenths of seconds. For more details, see the Vocabulary section in the Help menu.

1. MIDI is an acronym for Musical Instrument Digital Interface. It refers to a computer language established by electronic instrument manufacturers to enable computers and electronic musical devices (such as synthesizers) to communicate with each other. In a MIDI communication, each musical note is identified by a specific number. MicroWorlds Pro uses these same identification numbers for the notes.

Shortly, you will create several more turtles that make notes when you click on them. Before you do that, however, here is a project development tip to think about:

*Control actions with procedures where possible instead of placing commands into dialog boxes.*

During the development of any project, you will typically make several changes. It is usually easier and faster to edit procedures in the Procedures Tab area rather than changing commands in dialog boxes one at a time. This is especially true if the project has a number of turtles or other objects with instructions. Besides, the Instruction lines in dialog boxes do not have very much room for text.

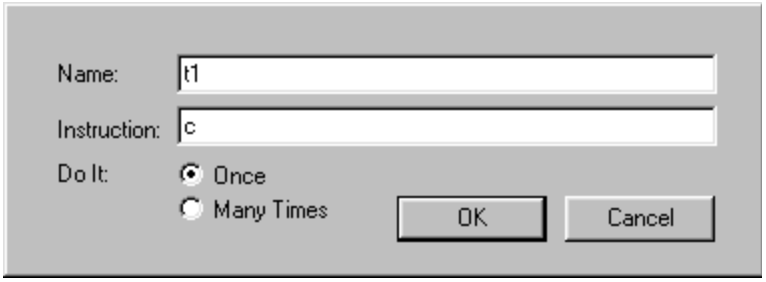
**Note**

Because MicroWorlds Pro is based on the Logo computer language, it is easy to make changes in a project at any time. Logo can accommodate a wide range of planning and development styles, from starting with a vague idea and messing around to working from a fully developed, detailed project plan. The example in this chapter is somewhere in between these two extremes.

Using the idea presented in the development tip above, make a procedure to play the middle C note by typing the following into the Procedures Tab area:

```
to c
note 60 4
end
```

Then change the turtle's instruction to **c**.




Click on the turtle to verify that it works.

Type the following additional procedures for other notes:

**Note**

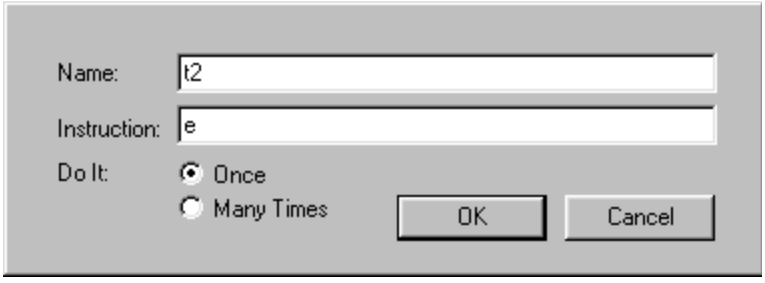
E above middle C has a note number of **64**, and G corresponds to note number **67** on the MIDI scale. Note number **72** is C but is an octave above middle C. Make sure there is no space between c and + in **c+**.

```
to e
note 64 4
end

to g
note 67 4
end

to c+
note 72 4
end
```

Now create three more turtles to go with your procedures.

Type a procedure name as the instruction in the dialog box of each turtle so that you have four turtles, each of which plays a different note when clicked. Here is a suggested assignment:

t1 **c**
t2 **e**
t3 **g**
t4 **c+**



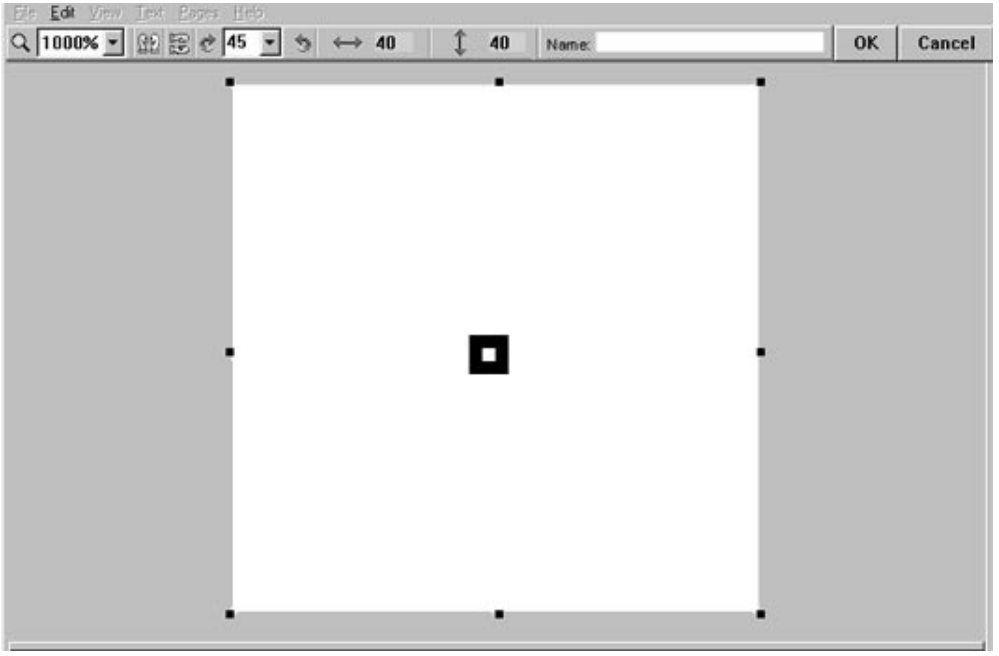
Remember to save your project before going any further.

Here's another project development tip:

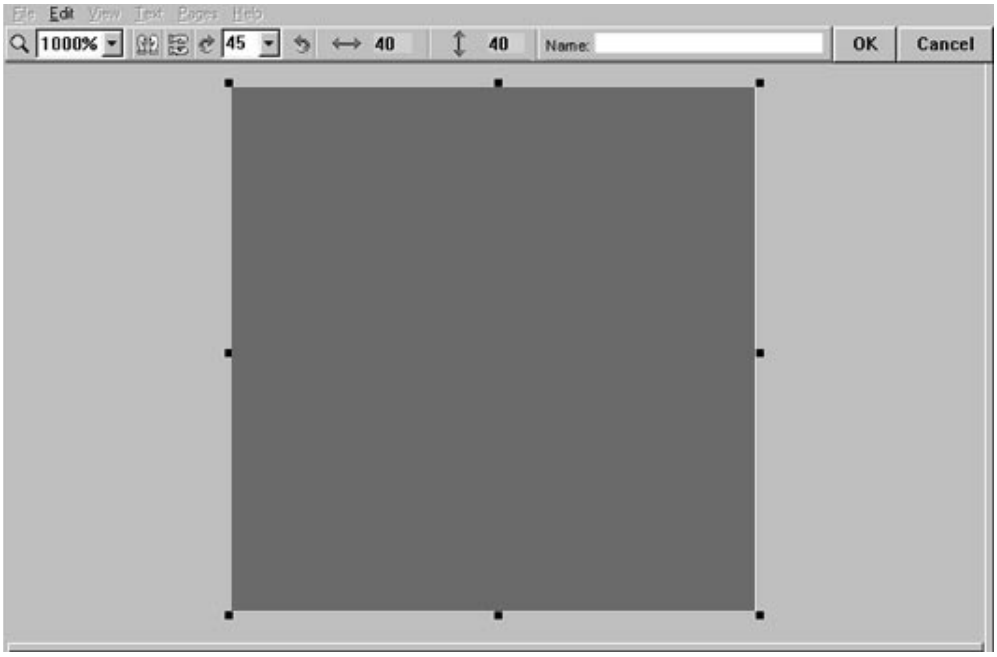
*Use the Save As feature regularly, and name your project files sequentially.*

For example, save your project initially as **project1a** then later as **project1b**, etc. This will give you a series of project files, each at a different stage of development. If you need to return to an earlier version, you will have it available.

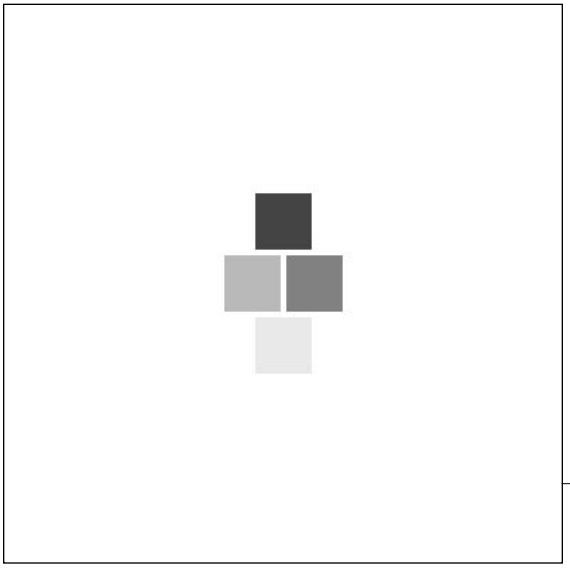
Now that you have some turtles that play notes when clicked, let's develop some colored shapes for them. Right-click Control-click one of the empty shapes in the Graphics Tab area and choose Edit to display the Shape Editor.



Use the Paint Can tool  to fill the empty shape with red. Name the shape **brightred**.




Repeat this process to make yellow, green, and blue square shapes named **brightyellow**, **brightgreen**, and **brightblue** respectively.



Extend the note assignments so that each turtle also has a different color associated with it. Change the shape of each turtle to the assigned colored square.

| | | |
|---|---|---|
| t1 | **c** | **brightred** |
| t2 | **e** | **brightyellow** |
| t3 | **g** | **brightgreen** |
| t4 | **c +** | **brightblue** |

Drag the square colored turtles into a four-cornered formation.

**Note**

While dragging the turtles, you might notice that one color seems to pass over another. This illustrates a concept called precedence. For objects such as turtles, the most recently created object has the highest precedence, and appears to pass over all others. The first turtle created has the lowest precedence and appears to pass beneath all others. You can also Right-click | Control-click on a turtle and choose In Front to assign precedence to it.

This is starting to look the way we want it to. When you click on a colored square-shaped turtle, it sounds its note. But it would be better if we could add a visual cue as well, perhaps a blink of some kind. One way to do this is to have the turtle change momentarily to a different shade of its color.

To do this, we will need four additional square shapes, each of a darker shade than the first four we made. Create these now and name them **darkred**, **darkyellow**, **darkgreen**, and **darkblue**.

Change all of the turtles to their respective darker colored shapes.

Now we need to change the procedures to include these colored shapes. Here is an example for the first one:

```
to c
t1,                          <- this line added
setshape "brightred          <- this line added
note 60 4
setshape "darkred            <- this line added
end
```

**Note**

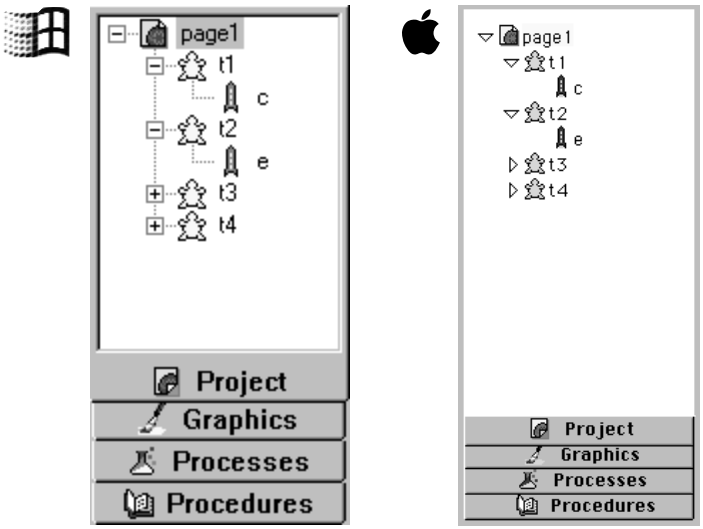
The turtle's name, **t1,** followed by a comma, directs the commands which follow to that turtle. In the line with **setshape "brightred** make sure you type the opening quotation marks immediately in front of the shape name. There are no closing quotation marks. MicroWorlds Pro uses the single set of quotation marks to indicate a special word. In this case, it is the name of a shape. Setting the shape to a bright color and then back to the dark color will make it appear to blink.

Now click on the red turtle and watch it blink as it sounds the note.

Change the **e**, **g**, and **c+** procedures for **t2**, **t3**, and **t4** in this same manner.

Then click on each of the turtles one at a time to verify that they blink as you want.

If you haven't saved your project in a while, this would be a good time to do so. Also, check out the structure of the project so far by clicking the Project tab. Click on any icon with a  to reveal more details.



Now that we have a set of turtles that play notes and blink when clicked, we need to set up a way to keep track of the notes.

A "player piano" idea can be used as part of the project. Let's make the computer remember a sequence of notes that you click and then play them back to you.

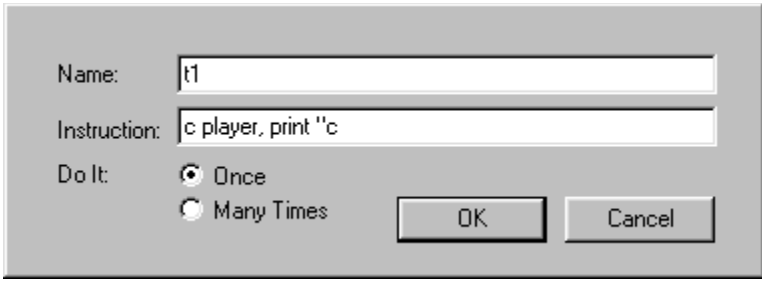
We can use a text box as a kind of *data structure* to store the names of the notes in sequence as you click the turtles. Then MicroWorlds Pro can read the names of the notes back out of the text box one at a time and run the procedures to play those notes in the same order. In this way, the text box functions something like the punched paper roll in a player piano. It stores the notes in sequence for later recall.

Create a tall, narrow text box on your project page and name it Player.

For MicroWorlds Pro to print the name of the note C in the Player text box each time you click on the red square, change the Instruction for the red turtle, **t1**, to the following:

c **player, print "c**

**this part added**




**Note**

The word **player** followed by a comma makes the Player text box active. The command **print "c** displays c in the Player text box. Make sure you include all spaces as shown.

Click on the red turtle to verify that MicroWorlds Pro prints **c** in the Player text box.

Change the instruction for the other turtles to the following:

yellow turtle: e **player, print "e**

green turtle: g **player, print "g**

blue turtle: c + **player, print "c +**

**these parts added**

Click on several of the turtles to print a series of different note names in the Player text box. The box is now functioning as a data structure to store the sequence of notes.

We need a way to clear the Player text box. Type the following procedure in the Procedures Tab area:

```
to clearplayer
player, cleartext
end
```

Now create a button with **clearplayer** as its instruction. Click on it to verify that it clears the Player text box.

Finally, let's work on playing the sequence of notes in the Player text box. With MicroWorlds Pro, it is amazingly simple. Since the note names are also the names of the procedures to play them, all we have to do is run the note names as if they were commands.

Create a button with **play** as its instruction and type in the following procedure:

```
to play
run player
end
```

Click on several of the turtles to create a list of note names in the Player text box. Then click on the **play** button to hear the notes being played back.

Experiment with some other sequences. Your "player piano" is complete!

**Note**

Although it appears fairly straightforward, this short procedure is quite sophisticated. We are using the **play** procedure to create and run another program! The **run** command treats the contents of the Player text box as instructions to carry out. In this case, the note names are also the names of procedures which sound the notes and blink the turtles. This process is possible because the **run** command requires a word or list as input. Although Player is the name of a text box, it is also a state variable which reports the contents of its text box. Here, it passes the sequence of note names to the **run** command.

## Round 3: Pick a Note

You have now developed two important parts of the interactive project. The set of turtles play the notes and blink when clicked. The Player text box stores the note names in sequence. The **play** button plays the sequence of notes back to you.

Next, we need to develop a way for the computer to pick notes randomly and keep track of them. This will make it possible for the computer to set up a growing sequence of notes for you to attempt to mimic.

Create another tall, narrow text box and name it Computer.


This is where MicroWorlds Pro can store the notes it randomly selects.

Define the following procedures to clear this text box and to run its contents, as you did for the Player text box:

```
to clearcomputer
computer, cleartext
end
```

```
to playcomputer
clearplayer
run computer
end
```

**Note**

These two procedures are similar to the **player** and **play** procedures. Note that **playcomputer** also includes **clearplayer**. This clears the Player text box before playing the notes listed in the Computer text box. Can you figure out why we suggest it be included here?

Create buttons to run the **clearcomputer** and **playcomputer** procedures.

Now comes one of the most interesting procedures in the entire project. Let's do a little preliminary work first. Type the following line in the Command Center:

```
show pick [c e g c+]
```

**Note**

Be sure to use square brackets. The **pick** command **pick**s at random one item from a list it is given as input. In this case, it picks a note name from our list of four notes. If you run this line again, it will likely select a different note, although it might choose the same one. For a better idea of this randomness, type **repeat 10 [show pick [c e g c+]]** in the Command Center.

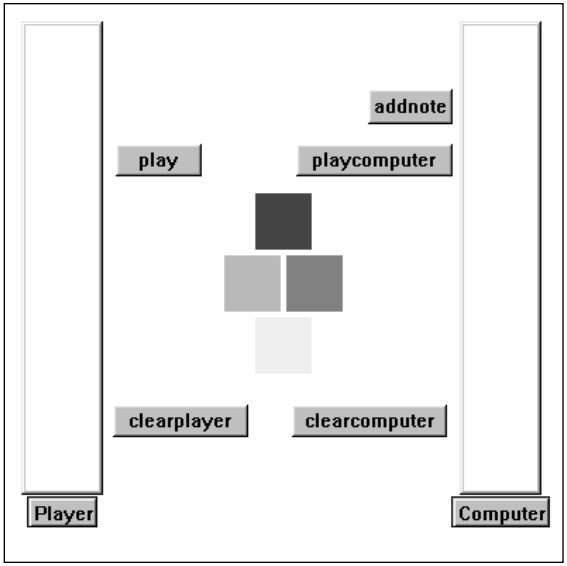
Now define a procedure to randomly pick a note:

```
to addnote
computer, print pick [c e g c+]
end
```

**Note**

In the **addnote** procedure, we have used **print** instead of **show**. The **show** command displays text in the Command Center; the **print** command is for use with text boxes. The word **computer** followed by a comma insures that the name of the randomly **pick**ed note is **print**ed in the Computer text box.

Create an **addnote** button, and then verify that all buttons work as expected.



Now you can begin to mimic the process of the interactive game. Follow these steps carefully. You will be writing a MicroWorlds Pro procedure shortly to do all of this and more!

1. Clear both text boxes by clicking the **clearcomputer** and **clearplayer** buttons.
2. Click on the **addnote** button. A random note is added to the Computer text box.
3. Click on the **playcomputer** button for MicroWorlds Pro to play the note(s) in the Computer text box.
4. Click on the colored turtle(s) to play the same note(s). You can see the note name(s) in the Player text box.
5. Compare the contents of the Player text box and the Computer text box. Are they the same? As long as you play the sequence back correctly, they are identical and you can continue. If they are different, it means you played the sequence incorrectly. The game is over for you!
6. Repeat steps 2 through 5 several more times, playing the sequence correctly.

This process gives you an idea how the game functions. As you can see, there is still some more work to be done. But you are making excellent progress!

## Round 4: Comparing Notes

At this stage, you have completed the backbone of your project. The only other things you need to do now are to set up a way for MicroWorlds Pro to compare the notes you play with those picked by the computer, report the results, and keep track of your performance.

It is probably not surprising to you that MicroWorlds Pro can perform step 5 in the sequence above. Making comparisons is something a computer can do easily and quickly. Let's explore one way to do that.

Create a button with **check** as its instruction. 

Add the following procedures:

```
to check
if not player = computer [oops stop]
ok
end

to oops
announce [Oops! You need more practice.]
end

to ok
announce [Your sequence is OK so far!]
end
```

Now you can play the game to any point you wish by clicking on the buttons and the turtles. When you are ready for MicroWorlds Pro to check whether the sequence you played is the same as that played by the computer, click on the **check** button.

As you are playing the game in this fashion, you might be wondering if the computer could automate everything so all you have to do is start the game and click on the colored turtles. That is exactly where we are headed next.

**Note**

The first line of the **check** procedure tells MicroWorlds Pro that, **if** the contents of the Player text box and the Computer text box are **not** identical, then run the **oops** procedure and then **stop**. This means that you have entered in an incorrect sequence, and the game is over.

The **if** command requires two inputs. The first input is a *conditional*, or a comparison that produces a report of either **true** or **false**. The second input is a *predicate*, or a list of instructions to be run if the conditional produces a result of **true**. The conditional here is **player = computer**. Since we want to run the **oops** procedure and **stop** the game if the contents of the text boxes are not equal, the output of **player = computer** is passed to **not**. Let's trace that process in detail.

Assuming that the two text boxes are different, the conditional **player = computer** outputs **false**. The **if** command runs the predicate only if the conditional is **true**. The **not** reporter outputs the logical inverse of its input. When it receives **false** from **player = computer**, it reports **true** to **if**. Then the **oops** procedure runs and the procedure **stop**s. Otherwise, the **ok** procedure runs. The **announce** command in the **oops** and **ok** procedures displays a dialog box containing the text in the square brackets.

## Round 5: All Together Now

At this point, you have developed a number of separate parts of the project, each of which does its job. Now it is time to write a *superprocedure*, that is, a procedure that calls (or runs) in sequence the procedures for the project parts you have developed. Most of the work is already done.

Just for fun, let's call the game **nomis** (SIMON spelled backwards), and use that name for the superprocedure. On the project page, create a button and name it **nomis**.

Add the following procedures:

**Note**

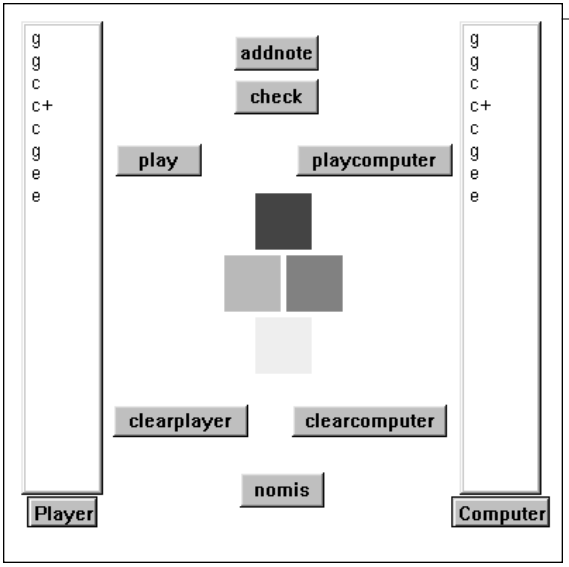
The tags show how the previous numbered steps are included in the procedures. Being able to "translate" from a step-by-step process to a group of procedures is an important skill to develop for effective programming and project development.

```
to nomis
clear
playgame
end

to clear
clearcomputer                          <- step 1
clearplayer
end

to playgame
addnote                                <- step 2
playcomputer                           <- step 3
yourturn                               <- step 4
if not computer = player [oops stop]   <- step 5
wait 5
playgame
end

to yourturn
waituntil [(textcount "computer) = (textcount
   "player)]
end
```

**Note**

The **nomis** superprocedure clears both text boxes with the **clear** procedure and then starts the game with the **playgame** procedure. **Playgame** follows the same sequence you outlined earlier. It adds a randomly selected note to the Computer text box and plays all notes in the box. Then it waits until **yourturn** is finished. In this case, the **waituntil** command in the **yourturn** procedure waits until both text boxes have the same number of note names in them. During this period of time, it is continuously counting the number of note names in each text box and comparing them. (See **textcount** in the Vocabulary section of the Help menu for details.) When you have clicked as many notes as the computer has in its list, then **yourturn** is finished. Next, **playgame** runs an instruction borrowed from the **check** button. As before, the **oops** procedure is run and the game **stop**s **if** the contents of the two text boxes are **not** identical. If they are the same, then the game continues. **Wait 5** is a brief pause.

Note that the final command of the **playgame** procedure is **playgame**. This is an example of *recursion*: the use of the name of a procedure inside the procedure itself. The procedure "runs a copy of itself," producing a repeating or looping pattern which is broken only when you complete a sequence of notes that does not match that of the computer. Every recursive procedure should contain a conditional to stop the procedure when a specified event occurs. Otherwise, the procedure runs forever! In such a case, you can click on the Stop All tool to interrupt a runaway recursive procedure. Recursion is an example of a simple but powerful feature of MicroWorlds Pro, and is a capability found in most higher level artificial intelligence and procedural computer languages.



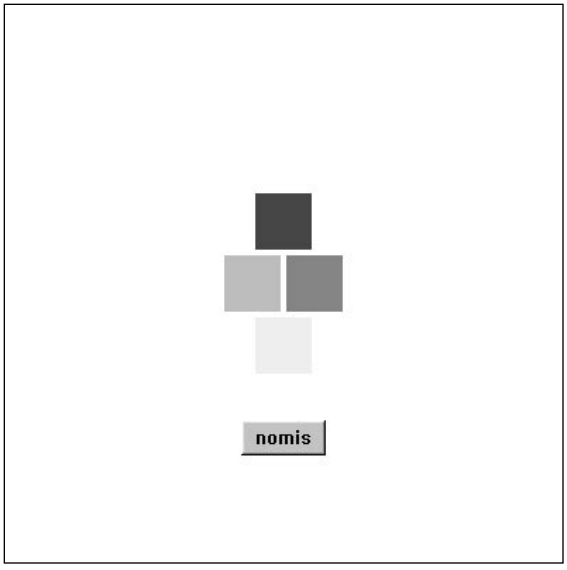

Click on the **nomis** button and try a round. Watch the action in the text boxes as the game is run automatically for you.

Now you can begin to add a finishing touch or two. Make the two text boxes invisible. Play the game once more to verify that it functions even if the boxes are not visible. Finally, select all of the buttons except **nomis** and click on the Cut tool  to eliminate them.

**Note**

In this project we have used *scaffolding*, a programming development technique in which the results of intermediate steps can be hidden or removed. In this case, the Player and Computer text boxes are hidden, but the game still employs them as data structures. You can always check in the Project Tab area to see the text boxes again if necessary. The buttons were removed because they were redundant. Each was included in one of the subprocedures called either by **nomis** or **playgame**.

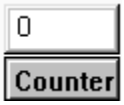
Try the game once more. Not bad!



## Round 6: Keeping Score

Finally, as a finishing touch, let's install a counter to keep track of the player's performance level.

Create a small text box and name it Counter.



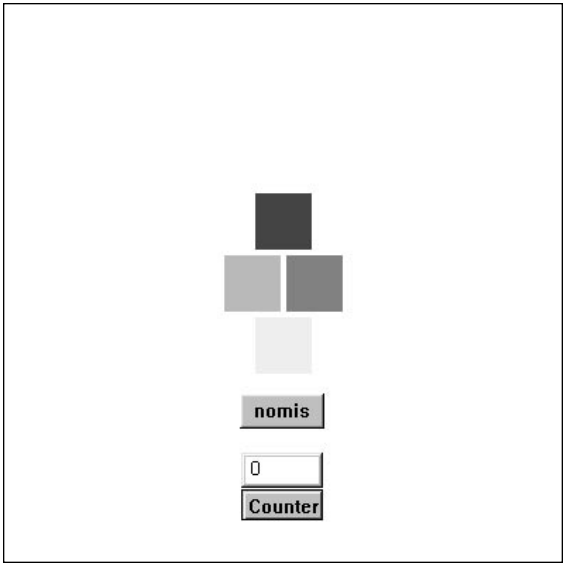

Change the **playgame** procedure and the **clear** procedure so they look like this:

```
to playgame
addnote
playcomputer
yourturn
if not computer = player [oops stop]
setcounter counter + 1          <- this line added
wait 5
playgame
end
```

```
to clear
setcounter 0                    <- this line added
clearcomputer
clearplayer
end
```

> **Note**
>
> The **setcounter 0** command in the **clear** procedure sets the value in the **counter** text box to **0**. **Setcounter counter + 1** increments this value by one each time the **playgame** procedure cycles successfully.

Now try out your interactive game. What is the highest level you can achieve?



## SUMMARY

This chapter was designed to introduce you to programming techniques and project development, and to acquaint you with the interactive features of MicroWorlds Pro. After developing the **nomis** musical game, you are now in a position to apply your programming skills and project development abilities in creating your own interactive projects. What ideas do you have?

Here are some possibilities to think about:

- Create a larger **nomis** game with more than four notes.
- Make each turtle move in a random fashion each time it is clicked during the game.
- Add a time limit to each turn or to the total length of the game. (HINT: See **timer** in the Vocabulary section of the Help menu.)
- Create a project variable to keep track of the highest score achieved (HINT: See **createprojectvar** in the Vocabulary section of the Help menu.)
- Create different turtle shapes from digital camera files, paint files, or other sources, and use them in the game.
- Redesign the **nomis** game to help you learn foreign language vocabulary. (HINT: Use the Microphone tool  to create sounds to use in the place of notes.)

For ideas on how to post your interactive project on the World Wide Web, see the following chapterette.

CHAPTERETTE B

# *Interaction Online!*

**NOW THAT you have learned a little about programming and project development and have designed an interactive MicroWorlds Pro project, why not share it with the world? This chapterette will teach you how to prepare your** nomis **or other MicroWorlds Pro project for uploading on the World Wide Web. Look at** Going Public on the Web **in the Tips and Tricks book for additional information.**

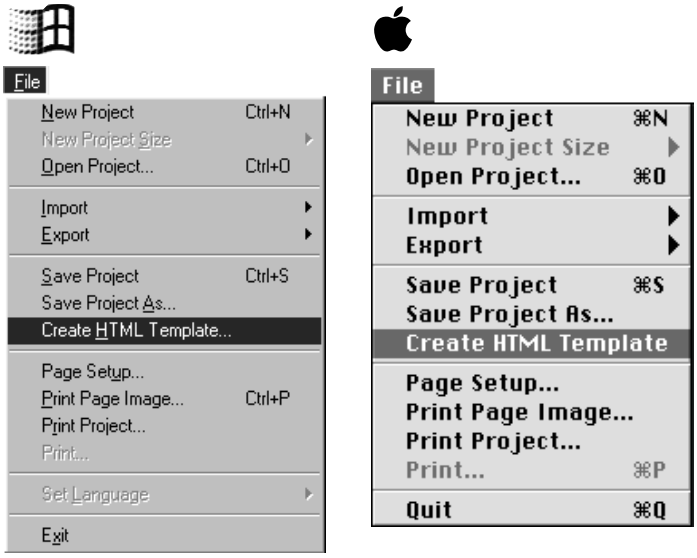
## Web Player Check

When you installed MicroWorlds Pro, the MicroWorlds Web Player is installed in the Plug-Ins folder of Internet Explorer. The Web Player allows you to view your MicroWorlds Pro project in your browser, prior to uploading it to the Web.

## CREATE HTML TEMPLATE

It's time to save your **nomis** project in a form appropriate for Web publishing.

Start up MicroWorlds Pro and open your **nomis** project. Then save it in web-compatible form using the Create HTML[2] Template selection from the File menu.




**Note**

**Note for Mac users**

For Mac users, if the current project does not have a mw2 or mwz extension, createhtml will first save a project with the mwz extension before creating the html template. Remember that it is the mwz project that is linked with the html page, not the one without any extension.

2. HTML is an acronym for HyperText Markup Language, the special computer language that makes it possible for computers of many kinds to access the resources of the Web. The HTML file can be run by your web browser using the Plug-Ins that you installed from your downloaded Web Player file.

## Take a Look!

Now you are ready to view your **nomis** project as someone might see it on the Web! Start up your Internet browser and use it to open your **nomis** HTML file.

That's all there is to it! You can operate your **nomis** project in your browser just as you did in MicroWorlds Pro. The important difference is that now others around the world will be able to experience it as well … as soon as you upload it!

Now, if you want to, you can add text to your project by opening it as an HTML file in your favorite HTML editor.

**Note**

When you open a project file in your browser, or when it is being viewed on the World Wide Web, the project runs in a type of presentation mode. This means that no one can edit or change it in any way.

## Loading Up!

The final step is to upload your **nomis** HTML file to the desired Web site according to the procedures given by your Internet service provider. Then create a link to it, and you are all set!

## In Conclusion

We hope you will enjoy sharing your interactive MicroWorlds Pro projects with others on the Web. From time to time, visit the LCSI web site at http://www.lcsi.ca to see examples of interactive projects from MicroWorlds.

CHAPTER 3

# *Investigating . . .*

## Overview and Objectives

**BY NOW, you are beginning to get the idea that MicroWorlds Pro is both powerful and easy to use. In this chapter you will learn more about the power of MicroWorlds Pro as a tool for investigation and for displaying results. You will create a project to help you learn more about these capabilities of MicroWorlds Pro. Along the way, you will learn how to design projects and write procedures that:**

- **set up data structures**
- **set up interactions with different data forms**
- **define interactions between data forms**
- **tabulate results of interactions**
- **graph results of interactions**

Change over time is one of the fundamental things that scientists and other curious people investigate. This helps us understand more about how systems function and what effects we might expect.

The investigative project you will develop is related to the work of Gregor Mendel, the man who discovered how traits are passed on from parents to offspring. Although Mendel's research dealt with pea plants, the results apply to many organisms, including humans.

First, a little background. When discussing the genetic aspect of a *phenotype*, or trait, textbooks typically use a combination of uppercase and lowercase letters to show the *genotype*, or genetic makeup. A dominant *allele*, or gene for a certain trait, is represented by a capital letter, and a recessive allele is represented by a lowercase letter. For example, in the phenotype of eye color in humans, the brown eye color (represented by B) is dominant over blue (represented by b).

A genotype for a trait is typically composed of two alleles. If at least one of the two is a dominant allele, then the dominant trait is expressed. If both alleles are recessive, then the recessive trait is expressed. Thus, people with genotypes of BB, Bb, and bB have brown eyes, but those with the bb genotype have blue eyes.

Either allele of a parent can be passed on to a child. A brown-eyed person with the Bb or bB genotype could pass on either allele to their child. However, a blue-eyed bb could only pass on the recessive b allele, and a brown-eyed BB could only contribute a dominant B allele.

Note: You might also encounter the terms *homozygous* and *heterozygous*. A BB genotype is classified as pure brown, and bb is pure blue. These pure genotypes are called *homozygous*. Bb or the equivalent bB would be called hybrid brown, or *heterozygous*.

For the purposes of this project, we'll use all uppercase letters for our genotypes. In particular, the capital A represents the dominant brown eye color and capital B represents the recessive blue color. This is because MicroWorlds Logo does not automatically differentiate between uppercase and lowercase letters and, therefore, would regard B and b as equivalent.

**How might two brown-eyed AB (or equivalent BA) parents pass these genes along to their offspring?**

This question will be the motivation for our initial investigation in this chapter.

During the fertilization process, half of the genetic material from each of the two parents is fused together. We can represent this process by joining an A or B allele from the first parent with an A or B allele from the second.

Here is a diagram of the four possible outcomes:

| | | parent1 | |
|---|---|---|---|
| | | **A** | **B** |
| parent2 | **A** | **AA** | **AB** |
| | **B** | **BA** | **BB** |

**Note**

Diagrams such as the above are sometimes called Punnett Squares.

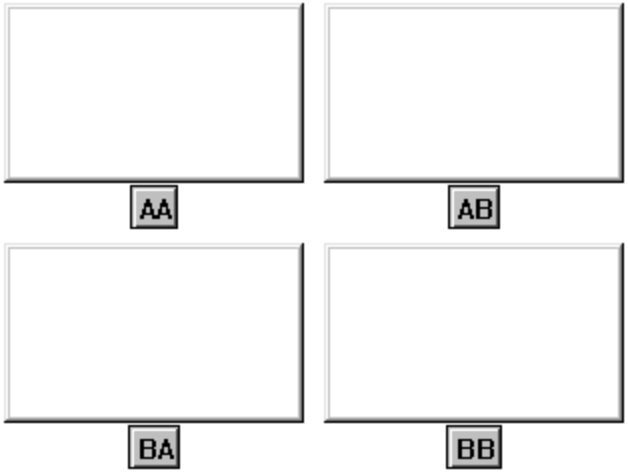
The offspring with the AA genotype will always have brown eyes, and will certainly contribute an A to a future offspring. The BB child will have blue eyes, and will likewise pass along a B to any future children. The AB and equivalent BA children will have brown eyes, but could pass along either an A or a B to their future children.

From this diagram, you can see the 1 / 2 / 1 relationship between pure brown, hybrid brown, and pure blue. This is the basis of the 25 / 50 / 25 percentile relationship which appears in discussions of Mendel's classic work.

Let's use MicroWorlds Pro to investigate this situation and see how close this ratio might be in practice.

## Genetic Clues

On the first page of a new project, set up four text boxes and name them AA, AB, BA, and BB respectively. Keep them the default size and move them to the upper left corner of the screen. Arrange them in the same order as in the table above. We will use these text boxes to keep track of the different combinations of A and B.




Type in the following procedure to initialize the text boxes:

```
to reset
setAA 0
setAB 0
setBA 0
setBB 0
end
```

**Note**

MicroWorlds Pro sets aside the name of each text box for use in a special **set** command. This command is constructed by combining "set" with the text box name into a single word. For example, **setAA 0** sets the contents of the AA text box to 0.

Now let's do a little background work to prepare for another procedure. Type the following in the Command Center:

**show pick "AB**

Either **A** or **B** is shown beneath the line you typed. Move the cursor back to the line and change it to the following,

**repeat 10 [show pick "AB]**

Now you have a listing containing both A's and B's. Odds are that there will be about the same number of each. This is how we will use MicroWorlds Pro to select an allele at random from a genotype.

**Note**

In Chapter 2, you used **pick** with a list of notes. Here, **pick** is given a word as input. It selects one letter at random and outputs it. In this instance, it **pick**s one of the letters of **AB** and reports it to **show**, which causes it to appear in the Command Center. Because it returns information, **pick** is called a reporter.

Now clear the Command Center with the **cc** command, and then type in the following:

**show word "A "B**

AB is shown in the Command Center. You can also use **show** and **word** to produce BA and other combinations. This is how we will use MicroWorlds Pro to fuse two alleles into a genotype.

**Note**

The **word** reporter outputs as a single word any two letters or words it receives as input. Note the use of the single quotation marks.

Now let's use this idea to create a procedure that will select one allele at random from each of two different parents and fuse them to make a child. Add the following procedure:

```
to child :parent1 :parent2
let [allele1 pick :parent1]
let [allele2 pick :parent2]
output word :allele1 :allele2
end
```

**Note**

**1** Note the two extra words **:parent1** and **:parent2** on the first line of the procedure. These words represent input variables to the procedure. A colon with a variable name represents the *value* of the variable. Throughout the procedure, **:parent1** will represent the value (genotype) of the first parent selected. **Let** creates a variable to be used within the procedure and then assigns a value to it. For example, the variable called **allele1** will have assigned to it a letter **pick**ed randomly from the genotype of the first parent given as input. **Output** sends out the name of the "child" created by **word** fusing the alleles from the two "parents."

**2** Those of you already familiar with the Logo computer language might wonder why **let** is used instead of **make**. After all, we might just as easily have written **make "allele1 pick :parent1**. In this instance, **let** demonstrates the use of a *local* (or temporary) variable. Any variable name created with **let** is used only within the procedure in which it is defined. This means that the memory used for the variable is free for other use after the procedure has ended. **Make**, on the other hand, creates a *global* variable which remains in memory and can be used in other procedures. (See Variables in MicroWorlds Topics for more information)

**3** In the MicroWorlds Pro vocabulary, there are two types of words. *Commands* carry out some action, and might require one or more input values. Examples include **clean**, **forward 100**, and **glide 50 5**. *Reporters*, on the other hand, output information of some kind. This means that there must be a command waiting to receive the result of a reporter. Otherwise, MicroWorlds Pro will not know what to do with the reported information. In the **child** procedure, **pick** is a reporter because it outputs a letter **pick**ed randomly from the name of the input parent. Because the **child** procedure also includes **output**, the procedure itself is a reporter.

Let's work with **child** a moment to see what it does. Type the following in the Command Center:

**show child "AB "AB**

A random genotype is shown.

> **Note**
>
> Note that the **child** reporter **output**s its result to the **show** command.

Let's look at some repeated results. Try the following:

**repeat 10 [show child "AB "AB]**

Examine the different genotypes that are generated. Most likely, you will see all combinations of the two letters, just as in the Punnett square earlier.

Now let's work on a way to keep an accurate count of everything. First, we need to see how the **set** command and the **get** reporter can work together.

The **set** command can be used in a general way to change the values of a variety of objects, including text boxes. Type the following in the Command Center:

**set "AA "text 225**

The **text** value in the AA text box is **set** to **225**.

> **Note**
>
> **Set** is an example of a very general command. Look it up in the Vocabulary section to see a listing of all the different objects it can affect. Here, we could have used **setAA 225** to accomplish the same specific result as we obtained. But once the **child** procedure begins functioning, we want to use the name of the child as the name of the text box whose contents we want to change. The **set** command is ideal for this situation.

The **get** reporter is useful in a similar way. Type the following in the Command Center:

**show get "AA "text**

The value of **225** is shown.

> **Note**
>
> The **get** reporter is similar to **set** in that it can be used with a variety of objects. See the Vocabulary section for more details. In this case, **get** is used to fetch the value of the **text** in the AA text box. It reports the value to **show** which displays it in the Command Center.

To keep track of an increasing number of AA genotypes, we will need to increment the number in the AA text box. Here is an example of how to use **get** do that. Type this in the Command Center:

**show get "AA "text + 1**

The resulting error message occurred because the **+** reporter tried to add the word **text** to the number **1**. In a case like this, parentheses are helpful; they force MicroWorlds Pro to **get** the value of the **text** in the AA text box first, and then add one.

**show (get "AA "text) + 1**

The result of **226** is displayed in the Command Center. You have incremented the former value of 225 by one. Now use both **set** and **get** to increment the contents of the AA text box itself:

**set "AA "text (get "AA "text) + 1**

> **Note**
>
> The current value of the AA text box (225) is fetched by **(get "AA "text)**, then increased by 1 and output as 226 to the **set** command. The actual instruction then carried out is **set "AA "text 226**.

These ideas can now be put together in a procedure to increment by one the contents of any designated text box. Such a general procedure is often called a *tool procedure*, because it can be used in a variety of situations. A tool procedure might accept one or more inputs; this allows you to use variables to specify where or on what the tool procedure operates.

The **increment** procedure below is an example of a tool procedure. It incorporates the lines you typed in the Command Center a few moments ago. Add this to your other procedures:

```
to increment :what
set :what "text (get :what "text) + 1
end
```

**Note**

Although **increment** is a short procedure, there is a lot of programming in it that you need to understand. Its main purpose is to set the value of the number in a text box to the current value plus one. Throughout the procedure, **:what** will represent the name of the text box with the value that we want to increase by one. As you saw earlier, the **get** reporter fetches the current count value in the **:what** text box. The **set** command changes the contents of the text box to the current value (as reported by **get**) plus one.

To verify that **increment** works, let's assume that **child** has just made a child with the AA genotype. Here is how **increment** updates the count of AA's. In the Command Center, type

```
reset
```

and then

```
increment "AA
```

You may wish to **increment** the count of other text boxes as well, just for fun. You have probably already observed that **increment** functions as a command and not as a reporter.

**Note**

The **generation** procedure is an example of a *superprocedure* that runs other procedures. The procedure called **child "AB "AB** randomly produces a child (either AA, AB, BA, or BB) from the two AB parents and outputs it. Here, this output is passed as *input* to **increment**. This illustrates how MicroWorlds evaluates all the input values to a command prior to running the command. Notice in **to increment :what**, the very first line of the **increment** procedure, that **:what** holds the place for the genotype to be incremented. The **increment** procedure then increases the count of the genotype of the child. This also demonstrates how powerful **child** is as a reporter because it can pass information along to other reporters or to commands.

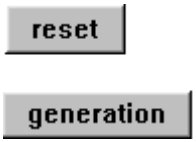
Finally, we need a procedure to get everything going:

```
to generation
increment child "AB "AB
end
```

Now let's finish the preparations for our investigation. Create two buttons, with **reset** and **generation** as their respective instructions. Drag the buttons to the lower left corner of the screen.

> **Note**
>
> You may have to lengthen the **generation** button a little. Select the button by dragging over it. Then drag one of the handles to adjust the size.

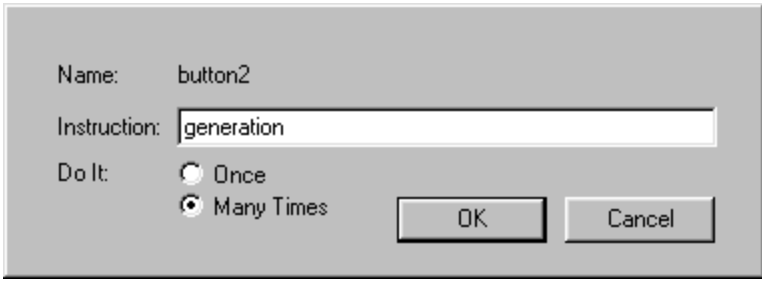
Click on the **reset** button to reset all genotype values to 0.




Now click once on the **generation** button.

The zero in one of the genotype text boxes changes to a one. Click on **generation** a few more times. Each time you click the **generation** button, it will increase the count of one of the text boxes, depending on the genotype of the offspring that is made from the alleles selected at random from the two parents.

Now the fun begins! Reset the values to zero with the **reset** button. Set the **generation** button to Many Times, and then click **generation**.




Wait until you have at least several hundred counts for each genotype, then click on the **generation** button once more to stop the process.

Observe the relationships among the counts. The numbers will not all be the same, but they are probably fairly close to one another. This means that you have an indication of the 1 / 2 / 1 relationship expected for pure brown / hybrid brown / pure blue. In terms of our original motivation, we are observing about three brown-eyed children (AA, AB, BA) for every blue-eyed one (BB).

> **Note**
>
> If you have not yet saved your work, this would be a good time to do so.

## Plot Twist

Let's add a counter to keep track of the number of generations. Create a new text box and name it Counter.


Make the following changes to the **reset** and **generation** procedures:

```
to reset
setAA 0
setAB 0
setBA 0
setBB 0
setcounter 0                    ← this line added
end

to generation
increment child "AB "AB
setcounter counter + 1          ← this line added
end
```

Click on the **reset** button, and then on **generation**. Observe how the **counter** value is continuously incremented!

Click on **generation** again to stop the process. The sum of the four numbers in the genotype boxes should equal the number in the Counter text box.

**Note**

If the sum of the genotype boxes differs from the **counter** by one, it probably means that you happened to stop the process between **increment**ing the **counter** and **increment**ing the genotype text box. This is nothing to worry about.

Now let's work on a way to display the results of our investigation. In addition to responding to commands such as **forward** and **back**, the turtle can also move using x- and y-coordinates (Cartesian coordinates). The middle of the screen is the Cartesian origin (0,0). Create a turtle so we can use it to plot our results in the upper right quadrant of the screen, since this is also where both x and y are positive.

Create a small text box and name it Plotwhat. This will be used to tell MicroWorlds Pro which genotype(s) you want plotted.


Type in the following procedures:

```
to plot
waituntil [counter > 0]
t1,
plotpoint counter (100 * (run plotwhat) / counter)
end

to plotpoint :x :y
setpos list :x :y
stamp
end
```

**Note**

The **plot** and **plotpoint** procedures are examples of tool procedures that we will be able to use in many ways. **Waituntil** makes sure that the **counter** is greater than zero. Then the turtle (**t1,**) is used to plot a point with the **plotpoint** procedure. In the very first line of **plotpoint** (i.e., **to plotpoint :x :y**), the **:x** holds a place for the value of the x-coordinate to be plotted, and the :y holds the place for the y-coordinate. When **plotpoint** is called within the **plot** procedure, that line also includes values for **:x** and **:y**. Specifically, in the line **plotpoint counter (100 * (run plotwhat) / counter)**, the **counter** reporter passes a value for the x-coordinate. The expression **(100 * (run plotwhat) / counter)** produces a scaled value for the y-coordinate, and may need a bit of explanation.

Plotting ratios and proportions is a widely-used technique in many types of investigations. In one such as ours, these proportions will give us information about the probabilities in effect. For example, we expect the number of AA's to be about 1/4 of the total, so a plot of **AA / counter** would verify this.

**Run plotwhat** reports the contents of the text box whose name is typed in Plotwhat. For example, if you had typed **AA** into the Plotwhat text box, then **(run plotwhat)** outputs the contents of the **AA** text box, that is, how many AA's there are at that moment. This **AA / counter** result of about 1/4 or 0.25 would not be very much distance for a y-coordinate on the page, so we need to scale it up; multiplying the **plotwhat / counter** proportion by 100 would do this nicely. **(100 * (run plotwhat) / counter)** is the value of the y-coordinate passed to **plotpoint**. A **list** of the **:x** and **:y** values is used as input to the **setpos** command to **set** the **pos**ition of the turtle to the location of the point to be plotted. **Stamp** does the rest.

Now let's add procedures to set up and draw the coordinate axes for the upper right quadrant:

```
to setup
make "origin [0 0]
make "height 150
make "width 250
end

to axes
t1,
pu setpos :origin pd
seth 0
fd :height bk :height
seth 90
fd :width bk :width
seth 0
pu
end
```

**Note**

A **setup** procedure with global variables such as **origin** provides flexibility. For example, the value of **origin** is made to be the list **[0 0]**, containing an x-coordinate value of 0 and a y-coordinate of 0. Feel free to modify the **height** and **width** values. You can test the **setup** procedure from the Command Center. In the **axes** procedure, **pu** stands for **p**en **u**p and allows the turtle to move without drawing a line. **Setpos :origin** moves the turtle to the location of the origin established in the **setup** procedure. The **pd** command enables the turtle to draw as it moves, by putting its **p**en **d**own. The **seth** command is used to **set** the **h**eading of the turtle to north (**0**) and to east (**90**) so that the x and y axes can be drawn using the **fd** (**forward**) and **bk** (**back**) commands. For more information on these commands, see the Vocabulary section in the Help menu.

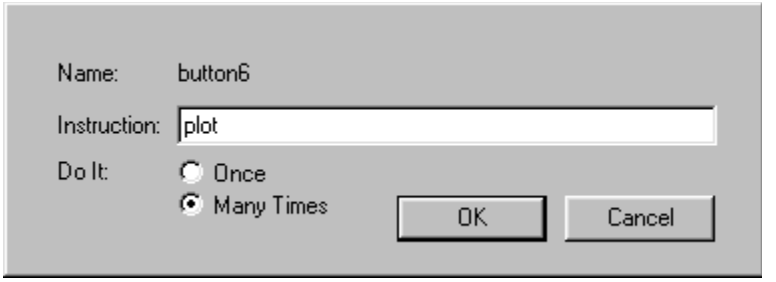
If you use a plotting origin other than **[0 0]**, you will have to offset the x- and y-coordinates by an appropriate amount.

Create buttons for **setup, axes**, **clean**, and **plot**.

**Note**

The **clean** command erases all the graphics (including **stamp**ed shapes) on the screen.

Set the **plot** button for Many Times.

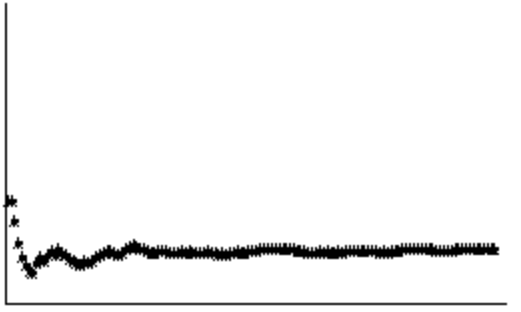



Click on the Demagnifier tool and then on the turtle three times. This gives a nice small shape to be stamped.

Now it is time to get started. Type **AA** in the Plotwhat text box.

Click the **reset** button to set all the genotype boxes and the Counter to zero. Click on **setup, axes**, and **plot**.

Then click on **generation**. When the plotted line nears the right side of the screen, click on **generation** to stop the process. Finally, click on **plot** to stop its process as well.



From the plotted results, you can see how the proportion of the AA genotype develops over time, approaching the expected value of 1/4. It varies greatly at the beginning, but rapidly settles down and approaches a value of (100 * 0.25) or a y-coordinate of 25. To check this, type the following in the Command Center:

**t1, show pos**

The first number shown is the x-coordinate (the value of the Counter), and the second is the y-coordinate. Most likely, the y-coordinate has a value close to 25.

Leave your plot of AA on the screen for now.

Before we carry our investigation further, let's condense some of the information. The AB and BA genotype are identical for all intents and purposes. Let us eliminate BA and combine its results with AB. Follow these steps.

1. Eliminate the BA text box by right-clicking on it and selecting Cut.
2. Delete the **setBA 0** line from the **reset** procedure, so it looks like this:

```
to reset
setAA 0
setAB 0
setBB 0
setcounter 0
end
```

3. Add a line to the **child** procedure so it looks like this:

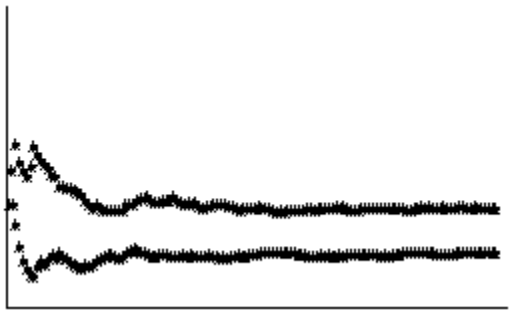
```
to child :parent1 :parent2
let [allele1 pick :parent1]
let [allele2 pick :parent2]
if :allele2 = "A [output word :allele2 :allele1]
output word :allele1 :allele2
end
```

this line added →

**Note**

The added line checks to see **if** the second allele is **A**, as would be the case with **BA** or **AA**. If this is true, then the procedure **output**s the alleles in the opposite order. **BA** would be output as **AB** which we want, and **AA** would not be affected.

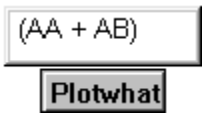
Now let's **plot** the proportion of the **AB** genotype and compare it to the **AA** plot. Type **AB** in the Plotwhat text box. Then click on **reset**, **plot**, and **generation**.



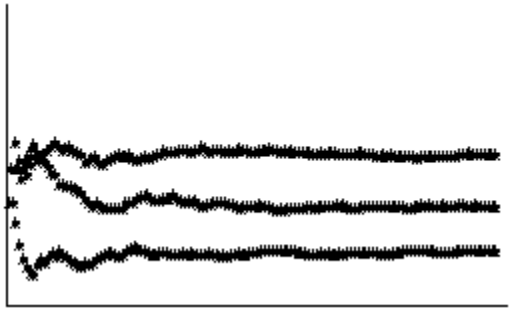
Click on **generation** and on **plot** to stop the process.

Note that the proportion value develops in the same manner as **AA**, but it is higher up on the screen. We are expecting an ultimate value of about 0.50. Use **t1, show pos** in the Command Center to see how close the y-coordinate is to this value. (The scaled value would be 100 * 0.50 = 50)

Keep both the **AA** and **AB** plots on the screen. Now plot the combined value of (**AA** + **AB**).




We expect it to have a proportion of about 0.75, so check for an ultimate y-coordinate on your plot close to 75.



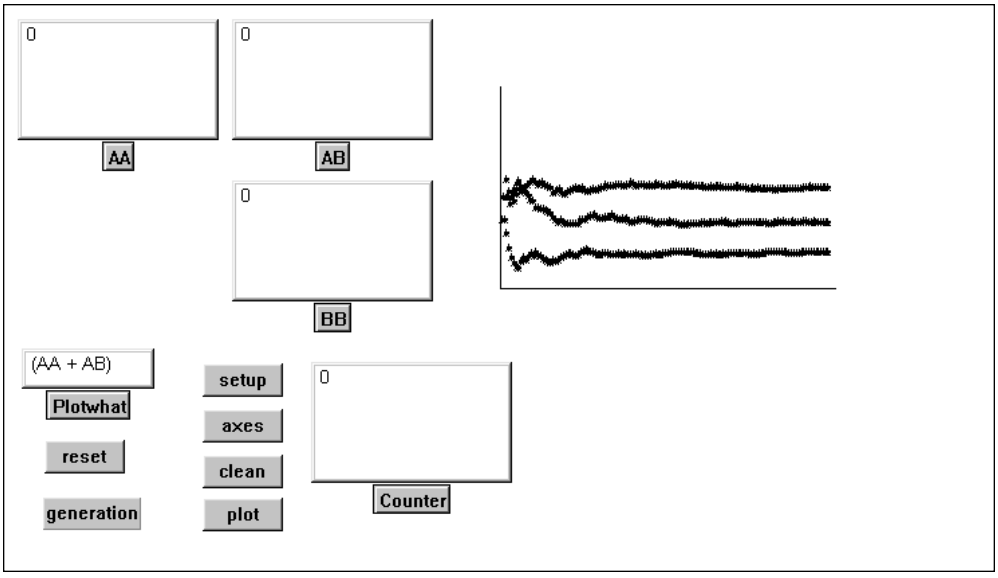
These investigative results confirm our expectations that two AB parents (with a dominant-recessive genotype for the eye color phenotype) will produce brown-eyed children about three-fourths of the time.

Save your project. You will make use of your work in the next section, in which we turn some things inside out!

## Inside Out Investigation

To this point, we have been investigating what might happen to the eye color of children with a single pair of known parents. Now let's turn our investigation inside out and set up a group of genotypes from which the parents can be selected randomly.

First, open the previous investigation project with the AB parents and choose Save As from the File menu. Save the project with a different name. In this way, we can build on your previous work.



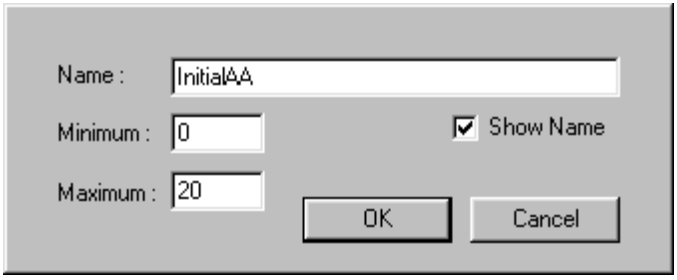

Investigation Question: How is the eye color distribution of a population affected by adding offspring to the population and by retiring adults from the population?

We need to make a few ground rules to keep the investigation manageable. You can add some variations later.

1. Two persons are selected at random from an established population to act as parents.
2. A child is made by selecting alleles at random from the selected parents, and is added to the population.
3. Each time a child is added to the population, a random person in the population (which now includes the child) is retired. This maintains the size of the population.

It might surprise you to learn that we will not have to make many changes from your earlier work. Here is what to do:

First, let's set up an easy way to establish an initial population of any proportion we want. Use the Slider tool to create three sliders. Name the sliders InitialAA, InitialBB, and InitialAB respectively. In the dialog box of each slider, set the maximum value to twenty. Leave the minimum value as zero.




**Note**

A slider is an object that can provide interactive control over a variable. With your mouse pointer, you can drag a slider arrow from left to right to produce any value between the minimum and the maximum shown. The current value of a slider is reported continuously by its name. You can use these sliders to set up your initial population conveniently.

Change the **reset** procedure to the following:

```
to reset
setAA initialAA        <- this line changed
setAB initialAB        <- this line changed
setBB initialBB        <- this line changed
setcounter 0
end
```

**Note**

The name of a slider acts as a reporter to pass the current slider value, in this case, to **setAA**.

Next, type in the following procedure to choose a person at random:

```
to chooseperson
let [num random (AA + AB + BB)]
if :num < AA [output "AA]
if :num < (AA + AB) [output "AB]
output "BB
end
```

You will also need the following decrement procedure to retire people from the population. (Compare this to the **increment** procedure from your earlier investigation.)

```
to decrement :what
set :what "text (get :what "text) - 1
end
```

**Note**

1. The **chooseperson** procedure selects a **random** number based on the total population size at that moment and assigns its value to the **num** variable. The procedure then tests the value of the variable (**:num**) and outputs a genotype based on the relative proportions of the population at that moment.

2. If it is not clear how the **chooseperson** procedure decides which genotype to output, perhaps this diagram will help. In programming, it is often more difficult to figure out the process of doing something than to write the actual procedures to do it.

| genotype population | AA | AA | AA | AA | AA | AB | AB | AB | AB | BB | BB | BB | BB | BB | BB | BB | BB |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| counting integers | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| random output | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |

**Note**

**Random** outputs a positive integer that is less than its input. As an example, **show random 2** would produce one of two numbers, either **0** or **1**, but not the number 2. In the **chooseperson** procedure, **random** outputs a number that is less than the total population. From that point, the procedure uses the current proportions of each genotype to determine the output genotype.

In the sample population shown above, the total count is 5 + 4 + 8 = 17 for AA + AB + BB. **Random (AA + AB + BB)** produces one of 17 numbers ranging from 0 through 16. Numbers 0 through 4 correspond to the 5 **AA** genotypes, 5 through 8 correspond to the 4 **AB** genotypes, and 9 through 16 correspond to the 8 **BB** genotypes. Thus, if the value of the **num** variable (**:num**) is less than 5, then the **AA** genotype is **output** and the procedure stops. Otherwise, if **:num** is less than 9, then its value must be in the 5 through 8 range; genotype **AB** is output and the procedure stops. These first two **if** statements will be bypassed only if **:num** is 9 or greater, in which the genotype **BB** is output.

In addition to the **chooseperson** and **decrement** procedures, we will need the following procedure to randomly select two parents from the population:

```
to chooseparents
let [parent1 chooseperson]
decrement :parent1
let [parent2 chooseperson]
increment :parent1
output sentence :parent1 :parent2
end
```

**Note**

The use of the **decrement** and **increment** procedure calls might seem confusing. However, they are essential to insure that the person selected to be the first parent is not also chosen as the second. In the last line of the procedure, the **sentence** reporter combines the two parent variables into a single list which it then passes to **output**.

Change the **generation** procedure as shown:

this line added to select two parents from the population

this line changed

this line added to keep the population the same size

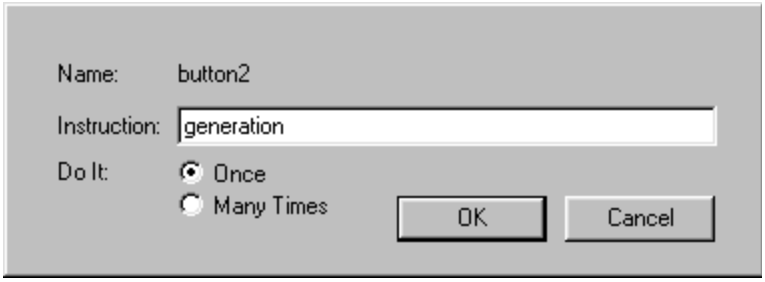
```
to generation
let [parents chooseparents]
increment child (item 1 :parents) (item 2 :parents)
decrement chooseperson
setcounter counter + 1
end
```

**Note**

**Let [parents chooseparents]** creates a local variable to temporarily store the list returned by **chooseparents**. When the **child** procedure is called, the two elements of this list, representing the selected parents, must be passed separately as inputs. **Item 1 :parents** returns the value of the first element in the **:parents** list and **item 2** functions in a similar manner. (Note: although the parentheses are unnecessary, they help to distinguish between the inputs.)

Set the InitialAA and InitialBB sliders to one and the InitialAB slider to zero. Click **reset**. This establishes a population of two, with one each of the AA and BB genotypes.

Change the Do It setting in the **generation** button's dialog box to Once.




Click the **generation** button. The AA and BB genotypes are selected as the parents, resulting in a child of genotype AB (brown eyes with one dominant and one recessive allele). At this point, the count for each genotype is one.

Then one of the three (AA, BB, and AB) is retired, as indicated by a zero in the box of that genotype.

If the AB genotype is still zero, then click successively on the **generation** button until the AB genotype has a count of one and either AA or BB is zero.

At this point, the child of the next **generation** will be made from alleles chosen randomly from the AB and either the AA or BB genotypes. For this example, let's assume that the homozygous parent has an AA genotype. A Punnett square shows the possibilities:

| | A | B |
|---|---|---|
| A | AA | AB |
| B | BA | BB |

If the child is of the AA genotype and the AB parent retires, then the entire population will consist of two AA genotypes, and no further changes can take place.

Click successively on the **generation** button. Sooner or later, either the AA or the BB genotype has a count of two, and there are no further changes.

This is an example of a phenomenon called "genetic drift," in which populations move toward certain terminal states. In this case, an initial population of one AA brown-eyed person and one BB blue-eyed person developed into a homogeneous population of either AA or BB genotypes.

Change the setting of the **generation** button to Many Times. Click on **reset** to establish a one-AA one-BB population once more, and observe the development at a slightly faster pace.

After you are satisfied that all such one-AA one-BB initial populations end up either as two blue- or two brown-eyed genotypes, then investigate the phenomenon with larger initial populations. For example, click on the sliders to set up an initial population of 3 AA and 3 BB genotypes. Does this population develop in a similar way? How long does it take?

The quantities of the AA, AB, and BB genotypes vary throughout the process. They do not seem to follow any particular pattern or tendency as the population develops. Plotting their numbers over time would show random-appearing behavior, with one of the pure genotypes eventually making up the entire population.

Our earlier plotting tools will not be helpful here. Right-click | Control-click on the **plot** button and the Plotwhat text box and select Cut to delete each one.

We can still use the **plotpoint** procedure, however, since it is very general in nature. Define the **scatterplot** procedure to display the relative number of AA's compared to BB's at any one time.

```
to scatterplot
t1,
plotpoint 5 * AA 5 * BB
end
```

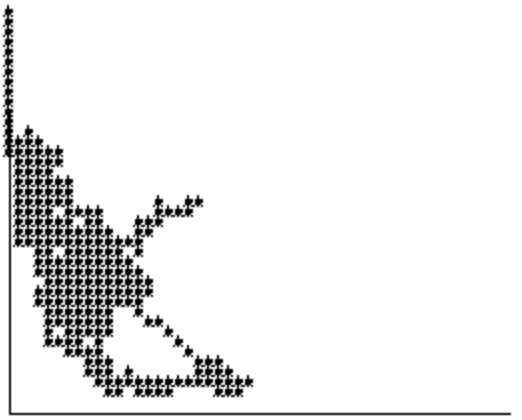
Then change the **generation** procedure as shown:

```
to generation
let [parents chooseparents]
increment child (item 1 :parents) (item 2 :parents)
decrement chooseperson
setcounter counter + 1
scatterplot                    <-- this line added
end
```

**Note**

The **scatterplot** procedure uses the earlier **plotpoint** procedure to plot the number of **AA** genotypes along the x-axis against the number of **BB** genotypes along the y-axis so that you can see the relative proportion of AA and BB genotypes in the population at any one time. The multiplier of 5 in **5 * AA 5 * BB** scales the plotting so that the small stamped turtles do not overlap. Placing **scatterplot** in the **generation** superprocedure makes sure that the status of the population is updated in each generation.

Set up an initial population and observe how it develops. You might start with 10 AA's and 10 BB's, then investigate others. Here is a sample result from an initial population of 20 AA's and 20 BB's:



Notice how the results of the plot appear to be distributed in a band with the general shape of a hyperbola. This behavior is typical of genetic drift situations.

With this plotting technique, you can also investigate development over time. Try this use of color with your investigation. Change the **plotpoint** and **axes** procedures as shown:

```
to plotpoint :x :y
setpos list :x :y
setc colorunder + 1          ← this line added
stamp
end
```

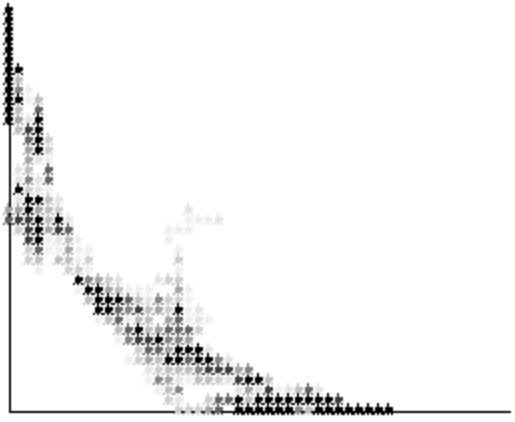
**Note**

Each time the turtle is now moved to a new location, it sets its color to a color with a number higher by one than the color of the location before stamping. This produces a definite color development on the screen. You may wish to look in the Graphics Tab area to display the sequence of colors while you are plotting your results.

Make the following changes to the **axes** procedure as well:

```
to axes
t1,
pu setpos :origin pd
setc "black                    ← this line added
seth 0
fd :height bk :height
seth 90
fd :width bk :width
seth 0
pu
end
```

Repeat a few of your earlier investigations. Now you can observe the color changes in your **scatterplot** as the turtle visits previously encountered states.



Save your project before going on.

## SCREAMING

You might have been wondering if there is any way to speed things up. In this section, we will do the same investigation as before, except we will use variables to scream or zip along. But watch closely, we will be using the variable names as variables themselves!

Save the previous project under a different project name. Use Cut to eliminate all text boxes. Edit the existing **reset**, **chooseperson**, **increment**, **decrement**, and **scatterplot** procedures as follows:

```
to reset
make "AA initialAA          <- this line changed
make "BB initialBB          <- this line changed
make "AB initialAB          <- this line changed
make "counter 0             <- this line changed
end
```

**Note**

The instruction **make "AA initialAA** creates a global variable whose value (**:AA**) initially carries the quantity given by the InitialAA slider. This variable and its value can be used by other procedures in the project. If we had used **let** instead of **make** to set up the variables, the values of those variables could not be used by other procedures. See the Variables section in Help Topics for more information.

```
to chooseperson
let [num random (:AA + :AB + :BB)]   <- this line changed
if :num < :AA [output "AA]           <- this line changed
if :num < (:AA + :AB) [output "AB]   <- this line changed
output "BB
end

to increment :what
make :what (thing :what) + 1         <- this line changed
end

to decrement :what
make :what (thing :what) - 1         <- this line changed
end
```

**Note**

The value of the variable **:what** in the **increment** and **decrement** procedures will be the name of another variable, such as AB. The instruction **thing :what** outputs the current value (e.g. the AA genotype count) of the variable named in **:what**. Let's go through an example in detail. Suppose we want to increment the AA genotype. **Make "x :x + 1** is the general form of the instruction line we are examining; it says to **make** the variable named x (**"x**) equal to the current value of the variable x (**:x**) increased by one. In our example, the input value for **:what** is the name **"AA**, so the instruction line must become **make "AA :AA + 1**. How can we get **:AA** from **"AA**? This comes from **thing "AA**. The expression **thing "AA** is equivalent to **:AA**. That is, **thing** outputs the value of the variable named AA.

```
to generation
let [parents chooseparents]
increment child (item 1 :parents) (item 2 :parents)
decrement chooseperson
increment "counter                    ← this line changed
scatterplot
end

to scatterplot
t1,
plotpoint 5 * :AA 5 * :BB             ← this line changed
end
```

Add the **info** procedure:

```
to info
setpopulation (se "AA :AA "AB :AB "BB :BB
   "counter :counter)
wait 2
end
```

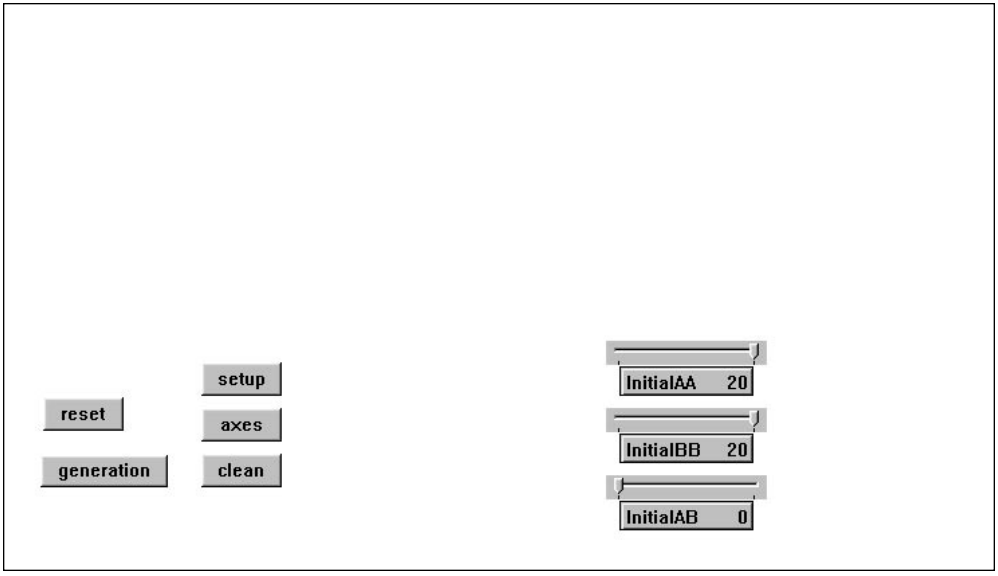
**Note**

The instruction **setpopulation (se "AA :AA "AB :AB "BB :BB "counter :counter)** sets the contents of the **population** text box (to be created) to a sentence made up of the word AA followed by the AA quantity, the word AB followed by its quantity, etc.

The following procedures remain as written:

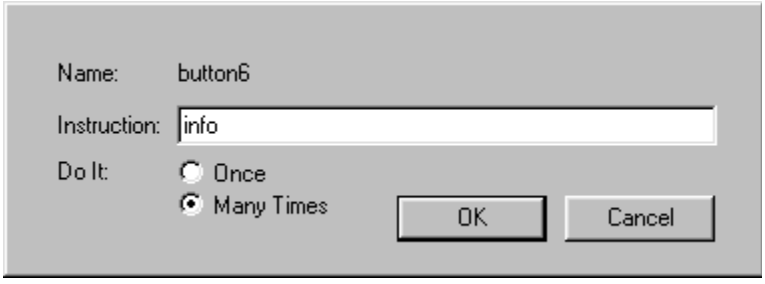
```
to child :parent1 :parent2
let [allele1 pick :parent1]
let [allele2 pick :parent2]
if :allele2 = "A [output word :allele2 :allele1]
output word :allele1 :allele2
end

to chooseparents
let [parent1 chooseperson]
decrement :parent1
let [parent2 chooseperson]
increment :parent2
output sentence :parent1 :parent2
end

to plotpoint :x :y
setpos list :x :y
setc colorunder + 1
stamp
end

to setup
make "origin [0 0]
make "height 150
make "width 250
end

to axes
t1,
pu setpos :origin pd
setc "black
seth 0
fd :height bk :height
seth 90
fd :width bk :width
seth 0
pu
end
```

In your project, you should have the **reset**, **clean**, **setup, axes**, and **generation** buttons, in addition to the three sliders named InitialAA, InitialBB, and InitialAB respectively.



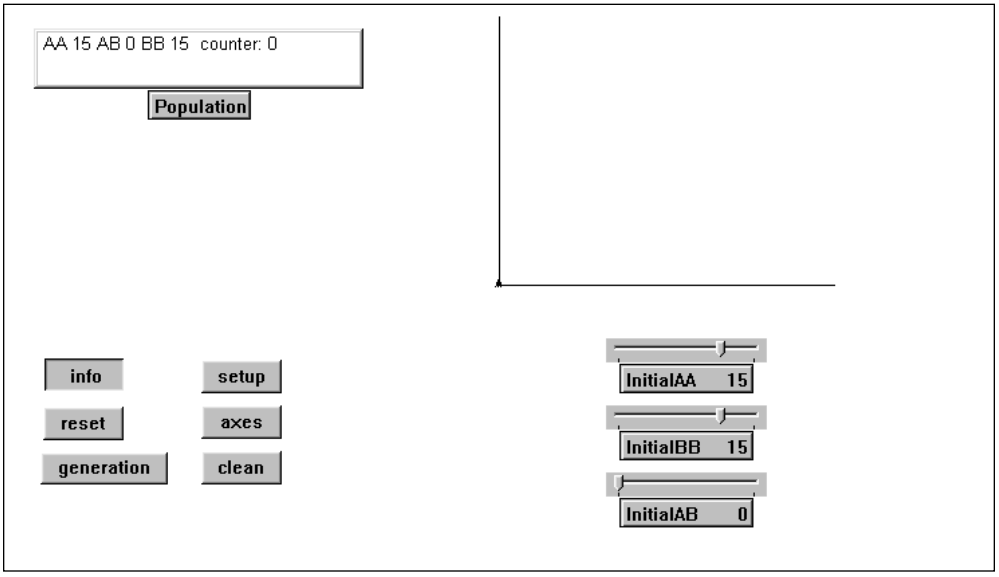

Create a short wide text box and name it Population.


Create a button with the instruction **info** set to Many Times.

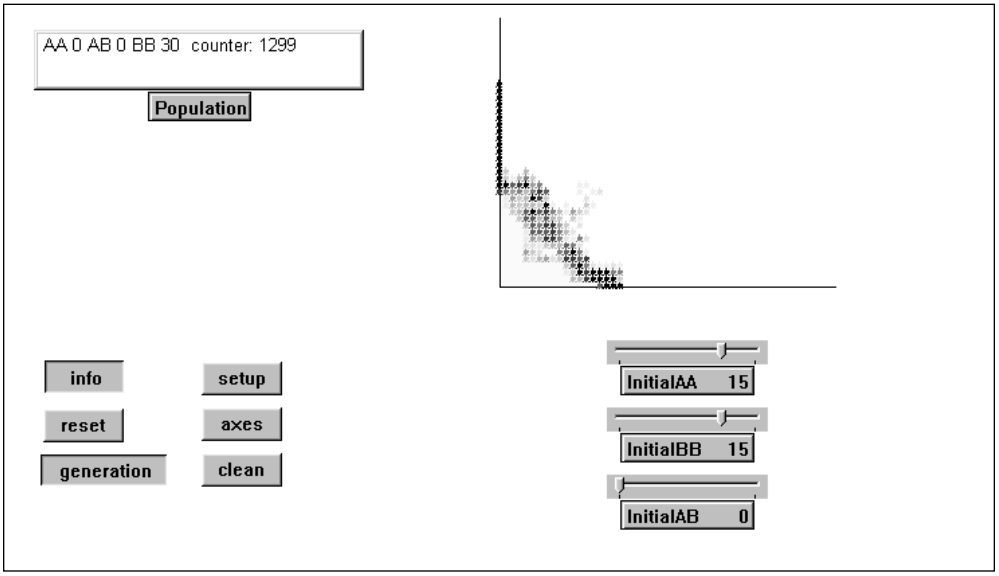


With the sliders, set up an initial population of 15 **AA's**, 15 **BB's**, and 0 **AB's**. Click **clean**, **setup, axes**, and **reset**.

Click **info** to display the population information.




Then click **generation** and watch the smoke!



Using variables instead of text boxes for data structures results in a significant increase in processing speed. Now you have the tools to conduct many more investigations of your own design.

**Note**

For some initial populations (i.e., 20 AA's and 20 BB's), the plotting might go off the top of the page. In this case, you could change the origin from [0 0] to [0 -50] and offset the plotted y-coordinates to keep everything on the page. To do this, make the changes shown below in the **setup** procedure. This creates two additional global variable whose values, **:xo** and **:yo**, need to be added to the x- and y-coordinates in **scatterplot**. The two procedures should then look like this:

```
to setup
make "origin [0 -50]
make "xo first :origin          <- this line added
make "yo last :origin           <- this line added
make "height 200                <- this line changed
make "width 250
end

to scatterplot
t1,
plotpoint :xo + 5 * :AA :yo + 5 * :BB   <- this line changed
end
```

## SUMMARY

This chapter was designed to introduce you to the powerful simulation and investigative features of MicroWorlds Pro. You are now in a position to design and carry out your own investigative projects. What ideas do you have?

Here are a few other questions you might find worthy of investigation:

- If you start with equal numbers of AA's and BB's, how many times out of 10 starts will the population develop into all AA's? all BB's? out of 100 starts? out of 1000 starts?
- If you start with **n** AA's ( **n** being an integer > 1) and one BB, how many times out of 10 starts will the population develop into all AA's? all BB's? out of 100 starts? out of 1000 starts?
- Repeat 2 above with n > 2 and two BB's. Then investigate higher combinations.
- Investigate initial populations with many AB's, and only a few AA's or BB's. Compare their development with populations from your earlier investigations.

Note: Investigations such as these which deal with probability and random events often require long periods of computer time. This means that the procedures must be run during times when computers are not being used for everyday tasks. One favorite technique is to run the projects overnight. MicroWorlds Pro is ideally suited for this type of investigation.

CHAPTERETTE C

# *It's All a Plot*

**You did some plotting with the genetics project. Let's take a look at how a more general plotting tool might be developed as part of another MicroWorlds Pro project.**

Many physical science experiments and investigations produce results that are measured over a period of time. For example, agricultural scientists might measure the height of an experimental hybrid plant each day to get an idea of its growth characteristics. Automotive engineers might use radar to measure the distance traveled every tenth of a second by a car to determine its acceleration performance.

Plotting the results of time studies such as these gives a graphical picture of the data as it changes over time, and often reveals aspects that are not easily derived by simply looking at the measurements.

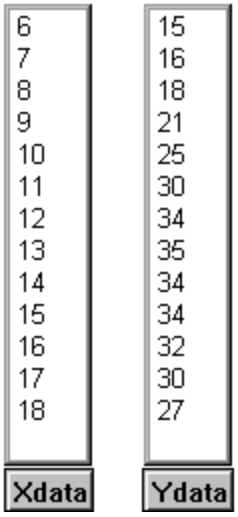
Typically, the first measurement is made at a specified moment, and the remaining measurements are made at regular intervals afterward. The total number of measurements or the length of the full time interval must also be specified.

To see how this kind of data is developed, let's take as an example the outdoor temperature being measured every hour during the day. At sunrise, you might record the temperature as 15 degrees Celsius.

Here is what your clipboard might look like for the whole day, from 6:00 to 18:00 (6:00 p.m.):

| Time | Temp (°C) |
|---|---|
| 6:00 | 15° |
| 7:00 | 16° |
| 8:00 | 18° |
| 9:00 | 21° |
| 10:00 | 25° |
| 11:00 | 30° |
| 12:00 | 34° |
| 13:00 | 35° |
| 14:00 | 34° |
| 15:00 | 34° |
| 16:00 | 32° |
| 17:00 | 30° |
| 18:00 | 27° |

The next task is to convert the 13 hourly temperature readings into a form that can be used by MicroWorlds Pro. One easy thing to do is to create text boxes named Xdata and Ydata and type the readings into them, pressing Enter after each value. Then make the boxes tall and narrow, and place them in the upper left corner of the page.



Now we can write procedures to plot our data. Plotdata is a new procedure, but plotpoint is an old friend from Chapter 3.

```
to plotdata :xlist :ylist
if empty? :ylist [stop]
plotpoint first :xlist first :ylist
plotdata butfirst :xlist butfirst :ylist
end

to plotpoint :x :y
setpos list :x :y
stamp
end
```

**Note**

1. The **plotdata** procedure is an example of a *recursive* procedure; it contains its own name as one of its instructions. You were introduced to this concept in Chapter 2. Here you see it used in a somewhat more complex manner. In this procedure, recursion causes the intervening instructions to be carried out repeatedly until a stopping condition is satisfied. In the case of **plotdata**, the plotting stops when all of the data has been plotted.

2. **Plotdata** needs two inputs: the lists of the x and y values to be plotted. If there is nothing in **:ylist**, the procedure **stop**s. Otherwise, it calls the **plotpoint** procedure and sends it the first item in the **:xlist** as the **:x** value and the first item in the **:ylist** as the :y value to be plotted. Then the procedure runs itself recursively, plotting the successive points until the list of y-coordinates is exhausted. In the recursive instruction, note that all data with the exception of the first element in each list is carried forward.

3. To take a detailed look at how recursion works in the **plotdata** procedure, use the **trace** function. See Programming Environment in Help Topics for details.

Experiment with this procedure. Create a turtle (if you do not have one), then in the Command Center, type

**plotdata parse xdata parse ydata**

**Note**

The Xdata and Ydata text boxes send out their contents as a string of characters. The **parse** reporter changes these characters into a list form that **plotdata** can use. Thus, the turtle shape is **stamp**ed by **plotdata** at points with x- and y-coordinates of (6, 15), (7, 16), (8, 18), etc.

Granted, this does not look like much right now. From this beginning, though, we will work on two refinements:

- Improve the appearance of the display by setting up the origin and axes, creating additional text boxes as labels, and changing the shape of the turtle.
- Scale the data points to fit the page.

Let's work on each in turn.

## Setting Up

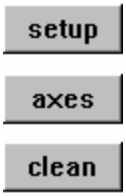
Let's have the plotting fill a good portion of the page. Here are procedures to set up variables for a plotting area 400 turtle steps wide and 250 steps high, with the origin in the lower left corner of the page:

```
to setup
make "origin [-200 -125]
make "xo first :origin
make "yo last :origin
make "height 250
make "width 400
end

to axes
pu setpos :origin pd
seth 0
fd :height bk :height
seth 90
fd :width bk :width
pu
end
```

**Note**

The **setup** procedure creates several global variables which can be used in other procedures. For example, **fd :height** in the **axes** procedure is using the value of the **:height** variable that was established in **setup**. The x- and y-coordinates of the origin, **:xo** and **:yo**, will be used later.

Create buttons for **setup** and **axes**. A **clean** button will also be useful. Stack these buttons in the lower left corner.




Try out your buttons.

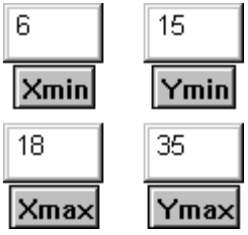
Create a shape for the turtle to use while plotting or use one of the bullet shapes.

## Scaling Down

In order for MicroWorlds Pro to plot data to a scale so it will fill the page area available, we need to know the maximum and minimum values of the data. One easy way to do this is to create text boxes for these values, which can be determined by inspection. Do this now and name the boxes Xmax, Xmin, Ymax, and Ymin.

**Note**

1. After you create the first small text box, you can select it by dragging, and then use the Copy and Paste functions to create three more to save time.
2. Of course, MicroWorlds Pro could find these values for you (and if you have long data lists, you might want to know about this option). The programming is somewhat complex, however. If you are interested, see the Tool Procedures supplement at the end of this chapterette.



Type in these procedures to scale your data:

```
to xscale :datum
output :xo + :width * (:datum - xmin) / (xmax -
   xmin)
end

to yscale :datum
output :yo + :height * (:datum - ymin) / (ymax -
   ymin)
end
```

**Note**

These procedures might look a little complicated, but they are based on a standard method of scaling data. Let's look at the **xscale** procedure in detail, realizing that **yscale** works in a similar way. Each data point, or **:datum**, must be subject to a proportion that takes into account the available plotting space (**:width**), the maximum and minimum values (**xmax** and **xmin**) to be plotted within that distance, and the offset distance of the origin from (0,0). The calculations in the parentheses produce the required proportional value. This is added to **:xo**, the x-displacement of the origin from x = 0. The result is a scaled value for the x-coordinate of the original data point that is appropriate for the size of the plot area.

Now revise the **plotdata** procedure to include these new tools:

```
to plotdata :xlist :ylist
if empty? :ylist [stop]
plotpoint xscale first :xlist                ← this line changed
   yscale first :ylist
plotdata butfirst :xlist butfirst :ylist
end
```

**Note**

**xscale first :xlist** outputs the scaled value of the first item in the data list for x and passes it to **plotpoint** as the **:x** value to be plotted.

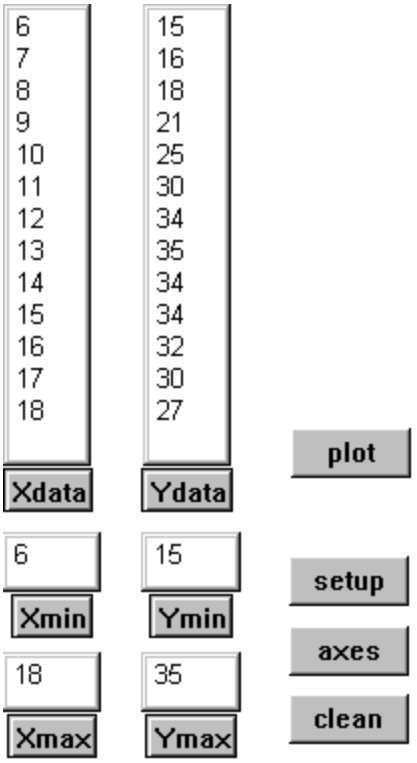
## The Plot Thickens

At last we are ready to put everything together! The idea here is to create a **plot** superprocedure that uses **plotdata** and the information in the various text boxes. Here is one that fits the bill. Part of it should look familiar, because we used it at the beginning of the chapterette!

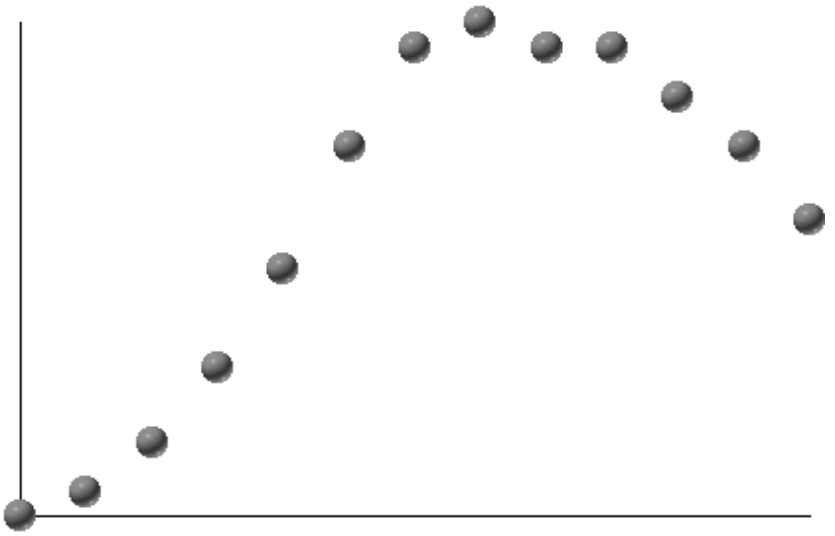
```
to plot
plotdata parse xdata parse ydata
end
```

Are you surprised that it looks so simple? This is because all of the difficult work was done earlier, making it possible to write such an "elegantly simple" superprocedure. But now you know that there is a lot of computing going on beneath the surface!

Add a **plot** button and you are ready to go!



Click on **setup**, **clean**, **axes**, and **plot** in that order.



Now try it with data of your own.

## Extras

As you might imagine, you have only begun to experience the potential of MicroWorlds Pro for displaying data from investigations. Here are a couple of additional ideas and suggestions you might want to try:

- Use color to plot your data in color.
- Plot several sets of data on the same page. You may want to use a different color and turtle shape for each plot. [Hint: Do not **clear** the page between plots. Type the new set of data into the text boxes and click on **plot**.]

And If You Really Want A Challenge

- Create a Function text box in which you type various functions or expressions (e.g. **3 * :x * :x + 4** ) to be plotted. Create sliders or text boxes for the initial and final values for **:x**, and for the size of **:xint**, the horizontal plotting increment.

## Maximum and Minimum Tool Procedures

As promised, here are some ideas on how to write tool procedures for MicroWorlds Pro to determine the maximum and minimum values of numbers in a list. The most useful form for this tool procedure to take is that of a reporter, as shown in the following procedures. The **maximum** procedure accepts a list of numbers (data) as input, and then **output**s (or reports) the maximum value in that list; the **minimum** procedure functions in a similar manner.

```
to maximum :datalist
make "what first :datalist
output max :what butfirst :datalist
end

to max :what :datalist
if empty? :datalist [output :what]
if (first :datalist) > :what [make "what first
   :datalist]
max :what butfirst :datalist
end

to minimum :datalist
make "what first :datalist
output min :what butfirst :datalist
end

to min :what :datalist
if empty? :datalist [output :what]
if (first :datalist) < :what [make "what first
   :datalist]
min :what butfirst :datalist
end
```

**Note**

**Maximum** and **max** work together in essentially the same way as **minimum** and **min**. For example, **maximum** first creates a variable named **what** and gives it initially the value of the **first** item in the data list. Then **maximum** calls the **max** procedure and passes along **:what** (the *value* of the **what** variable) and all **but** the **first** item of the data list as input values.

**Max** compares the value of the **first** item in its input data list to the current **:what** value. If it is greater, then the procedure **make**s it the new value of the **what** variable. **Max** continues recursively with all **but** the **first** item of the data list until all data have been examined (that is, until the data list is empty). At that point, **:what** contains the maximum value in the data list and is reported as **output** to **maximum**, which **output**s it in turn.

# *Where To Go From Here?*

**In the preceding three chapters and chapterettes, you have glimpsed only a little of the potential of MicroWorlds Pro. You learned some techniques in Chapter 1, developed programming and project development skills in Chapter 2, and gained experience investigating and displaying data in Chapter 3. As every journey begins with a single step, you have now just begun your journey into the exciting world of multimedia, programming, and Web publishing! Think of the possibilities!**

Let's revisit the question posed in the Preface. How might you use MicroWorlds Pro? You can go in just about any direction you can imagine. Here are just a few suggestions for additional activities and project ideas. And don't forget: you can post many of these on the Web as well!

- Use MicroWorlds Pro to analyze MicroWorlds projects that have already been developed. The tree display at the Project tab can give you many insights and new perspectives on project design.
- Explore the Processes tab to track processes that are running in the background. For example, see Programming Environment in the Help Topics.
- Develop some projects of your own, such as:
  - Brief interactive tutorial to highlight specific content or review, or to teach a specific skill.
  - Interactive student files, with digital photographs, sound bites, etc.

- Foreign language vocabulary builder.
- Database of scanned artwork.
- Musical puzzles.
- Interactive games.
- Simulations.
- Random events (drawing, music, colors, etc.).
- Mazes.
- Explorations of motion, including effects of gravity, friction, inertia, wind, etc.
- Card games.

The Tips and Tricks book provides many ideas as well as useful information for getting around MicroWorlds. Look at the Help Topics, especially the Logo Programming section, for more in-depth discussion on programming ideas.

Finally, don't forget to visit the LCSI web site frequently at http://www.lcsi.ca for product updates and other project ideas.