7

Chapter Seven
MAKING OBJECTS MOVE

In earlier chapters, we've learned how to create Objects, place them at a point on the screen, destroy them, and manipulate their color.

Now it's time to learn how to get Objects to move smoothly on the screen.

Motion: Guided vs. Unguided

Proof supports two types of motion: unguided and guided.

Unguided motion occurs along a straight line between two points. It is useful for moving an Object to a destination in cases where the exact destination location is not known when the layout is built but can be calculated while the model runs. One example would be a frontal view of a vertical storage/retrieval system with many randomly accessible row and column destinations. Another example might be free-ranging vehicles in a large warehouse. Another example is a communications network in which packets are transmitted between pairs of nodes from among a large set of nodes.

Guided motion takes place along a Path defined in the Layout. Guided motion is useful for movement that corresponds to a physical guide Path (*e.g.* a conveyor or a railroad), for displaying any type of physical or logical accumulating queue (*e.g.* message requests on a computer network), and even for data-driven movement through a network of Paths. Chapter 12 contains a full discussion of defining Paths.

Using Paths for movement is often the preferred approach. Because Paths are referenced by name in the trace file, the underlying model does not need to know exact coordinates of points in the animation. Paths are also more flexible because they can include curves and can perform automatic accumulation and other automatic functions.

We'll start our discussion of Object motion by covering unguided motion using the move command. After that, we'll move on to guided motion using Paths.

Moving Objects

The move command causes an Object to move in a straight line between two coordinate points. The syntax of the move command is either of the following:

move *objectID duration xdest ydest* [**relative**]

move *objectID* **speed** *s xdest ydest* [**relative**]

The *duration* is a value that is specified in units of animated time. The speed at which the Object will move is implied by the duration and the distance between current and destination coordinates.

Instead of the *duration*, you can use the optional keyword speed and specify *s*, the speed. The speed is distance per time unit at which the Object will travel. In this case the duration value is determined by the speed and the distance.

The Object will begin traveling from wherever it currently is toward the destination point (*xdest, ydest*) immediately. Assuming no other trace command tells the Object to do something else along the way, the Object will come to rest at the destination point after the exact duration value has elapsed since the start of the move

If the optional relative keyword is used, the Object begins moving from its current location toward a point that is *xdest* units away in the x direction and *ydest* units away in the y direction. For example:

```
move 21 speed 10 4 6 relative
```

Here Object 21 will begin to move 4 units in the positive x direction and 6 units in the positive y direction from its current location traveling with a speed of 10 units/time.

The Object will always move in a straight line *from its current location*. You can't move an Object before it has been placed at least once. Therefore it will always have a current location. Trying to move an Object that has no location results in an error.

Consider another example of move. Assume that Object "Car" has already been created and placed at some (x, y) point.

```
time 50
move Car 30 40 50
time 90
move Car speed 1 3 4 relative
time 100
```

At time 50 several things happen. Proof determines the distance to (40, 50) from the Object's present location, then divides the distance by 30 to determine a speed. This internally calculated speed is used to determine how far the Object will move at each screen update. Proof also schedules an internal event at time 80 (because 50 + 30 = 80) to turn off the motion.

Proof spends the interval between time 50 and time 80 moving the Object smoothly across the screen (and carrying out any other animation trace commands not related to this Object). At time 80, the Car stops at location (40, 50). Proof has not read the trace file line following time 90, and won't do so until the animated time reaches 90.

At time 90, the Car begins to move to (43, 54), which happens to be exactly 5 units away along the diagonal, with a speed of 1 distance unit per time unit. It will reach its destination at time 95 (5 distance units at speed 1 is 5 time units; 90 + 5 = 95).

Exercise 7-1: Moving an Object Using the Move Command

Let's try three examples that use move. We'll continue to rely on shapes.lay (used previously in Exercise 6-3) for our Class definitions).

First, using your text editor or word processor, create a new text file called mymove.atf that moves a single Object using roughly the same direction and speed used in Exercise 6-3. What previously took many lines of trace file commands now takes just one line, so your trace file will need only three commands: create, move, and end.

In order to formulate your move command, you have to know the duration of the move. To determine the duration, you can first determine the distance (in units in the Proof coordinate space) and the speed (in coordinate units per unit of animated time) based on your solution to Exercise 6-3. Based on that distance and speed, you can calculate the duration.

After saving mymove.atf, run the animation as follows:

1. Choose **File**, **Open Layout**, and select shapes.lay.
2. Choose **File**, **Open Trace**; and select mymove.atf.

3. Click the "running man" icon on the toolbar to start the animation.

For the second example, try setting up the following scenario by modifying mymove.atf. A single Object will begin moving at time 15 from (x=10, y=35) to (x=60, y=25) following the diagram in Figure 7-1. (Your screen will not show any of the markings in the diagram, because you are still using shapes.lay, which has nothing drawn in it.) Try using the same duration for each of the move commands. What happened to the Object's speed?

Now, as the third example, try maintaining the Object's speed at 5.0 coordinate units per animated time unit throughout its journey while visiting each corner depicted in the following diagram. You'll have to add the speed keyword to your move commands to complete this.

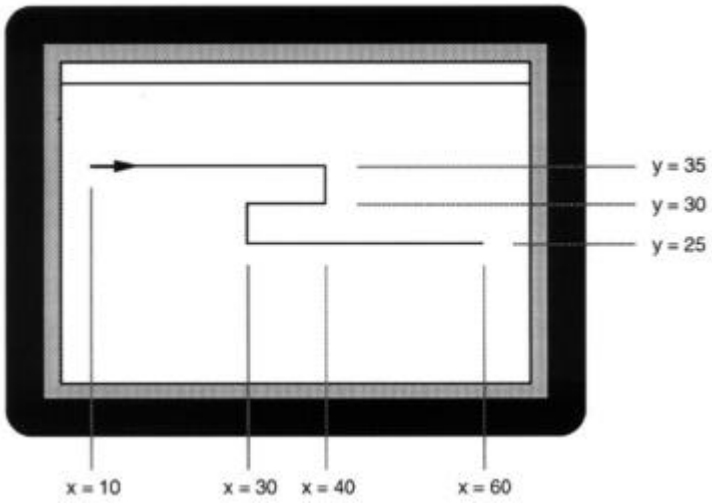

Figure 7-1. Unguided motion. Use the move command to move your object as indicated by this diagram.

Some example animations, random and burst (located in the sample folder), show other ways you can use move for unguided moves. (If you run them, be sure to use **File, Open Layout+Trace** and not just **File, Open Trace**, because they use different layout files.) While move is useful for special effects such as these, you will probably use Paths for most of your animation work.

Although it is not prohibited, you should normally avoid mixing move commands and Path-related commands for a given Object. Generally only one of the two approaches will "fit" a given situation.

Moving Objects Along Paths

In this section we will discuss simple use of Paths with examples that contain pre-defined Paths. Path definition is covered in Chapter 12.

These are the main characteristics of Paths:

- A Path can have any combination of straight and curved segments.
- Each Path has a beginning point and an end point.
- Normally a path is continuous, but it can be *disjoint* (can have gaps)
- Each Path has a speed, which you can specify in various ways.

What does a Path do? If an Object is placed onto a Path and the Path speed or Object speed is greater than zero, it will automatically move along the Path until it reaches the end, unless some other animation trace file command (e.g., destroy or end, or even a different place instruction or a move) interrupts its motion.

The command for using Paths is place…on. Because there are underlying data structures containing speed and geometry information, place…on is less complicated than move. The syntax of place…on is:

place objectID **on** pathname

Here are two examples of the place…on command.

```
place 1 on ThePath
```

```
place Box on Conveyor
```

This command causes Proof to place the Object onto the specified Path at the beginning point of the Path. The Object will immediately begin moving along the Path from Segment to Segment until it reaches the end (or accumulates – see below), or until a subsequent animation trace file command interrupts its motion.

Exercise 7-2: Moving an Object Along a Path

A Path named "ThePath" is defined in the supplied file path.lay (located in the exercise folder). Its path speed is 1.0. Its geometry is very similar to the "path" you used in the move example (see Exercise 7-1).

Edit a new file, mypath.atf, that places an Object on ThePath at time 15. (The shapes from shapes.lay, used in previous exercises, are also included in path.lay.)

Remember, when working with short sample trace files, to use at least one time command to allow the animation to run long enough to see the results of your other commands.

Open the animation by choosing **File**, **Open Layout**, and selecting path.lay, then choosing **File**, **Open Trace**, and selecting mypath.atf. Run the animation.

Once again, you've replaced a string of trace file commands with a single command. You probably won't want to use move in situations that resemble this one, unless it is impractical to define Paths in advance.

Unless you specify otherwise, the Object will move at the speed associated with the Path (as defined in the layout). The units of speed are linear units in the Proof coordinate space per unit of animated time. (Dynamically changing Object speeds and Path speeds using animation trace commands is possible. These advanced topics are discussed in Chapter 14.)

When the Object reaches the end of the Path, it stops moving but remains visible on the screen. (An exception occurs if the Path is "circular." Whether or not a Path is circular is defined in the layout. Circular Paths are discussed in Chapter 12.)

Try creating more Objects and placing them on "ThePath at different times. What happens to them when they reach the end of the Path

Below is a sample version of the mypath.atf file.

```
time 0
create TRIANGLE 1
time 15
place 1 on ThePath
time 20
create SQUARE 2
create SQUARE 3
place 3 on ThePath
time 30
```

```
place 2 on ThePath
time 100
end
```

Accumulating Paths

Paths can be "accumulating" or "non-accumulating." Proof will temporarily stop (or slow down) an Object before the end of an *accumulating* Path if it encounters another Object that is stopped (or moving more slowly) on the same Path. When the blockage goes away, the Object automatically resumes its motion until it reaches the end of the Path or is blocked again.

If a Path is not accumulating, then multiple Objects that congregate at the end of the Path will come to rest on top of one another.

Exercise 7-3: Accumulating vs. Non-Accumulating Paths

The file apath.lay (located in the exercise folder) contains "ThePath" and "TheAccPath." TheAccPath is similar to ThePath but is an accumulating Path. Our familiar shapes from shapes.lay and path.lay are again available. The Paths are identified on the screen.

Let's experiment with accumulating vs. non-accumulating Paths by modifying mypath.atf (from Exercise 7-2) so that it places Objects on either Path ("ThePath" or "TheAccPath") at various times. (If you did not create the mypath.atf file in Exercise 7-2, you can access a completed version from the sample folder.)

Once you've finished and saved the changes to mypath.atf, go back to Proof and run the animation by opening apath.lay and then mypath.atf.