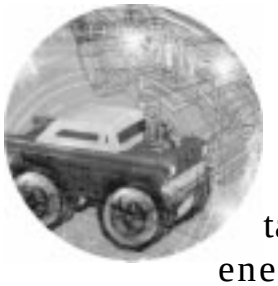
15

TUBE CLEANER: A SIMPLE SHOOTING GAME



Tube Cleaner was designed by Freid Lachnowicz. It is a simple shooter game that takes place in a tube. There are three kinds of enemies, and your goal is to collect as many points and bullets as you can as you travel up the tube. (To play the finished game, you can load the scene Games/Tube-Cleaner.blend from the CD. Instructions are included.)

Here's a look at the opening screen:

This tutorial will not to teach you how to create your own version of Tube Cleaner from scratch. Instead, it will teach you how to create interactivity. And, of course, you are encouraged to change and extend the game to your liking!

The Tube Cleaner game controls are rather simple:

Table 15.1: Tube Cleaner Game Controls

Controls	Description
Arrow keys	Rotate the cannon left, right, up, and down
Spacebar	Shoot

15.1. Loading the Models

To begin this tutorial, start Blender and load Tutorials/TubeCleaner/TubeCleaner_00.blend from the CD. This scene contains all of the models that you'll need to start with. We'll show you how to make the scene interactive by showing you how to:

- Add game logic to the gun, allowing it to move up, turn, and shoot
- Add game logic to the enemies
- Create the score system including a display
- Provide extra bullets

As shown in Figure 15-1, the scene contains a CameraView on the left, a wireframe view (view from top, TopView) on the right, and the RealtimeButtons on the bottom. In the TopView you can see that the Base object is already selected and active (it is purple in the wireframe).

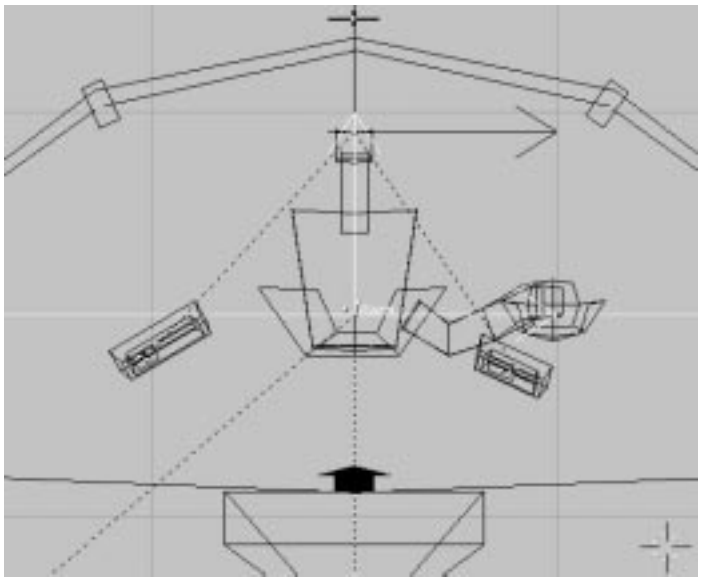


Figure 15-1: Wireframe TopView in the loaded Tube Cleaner scene

The Base object will carry the gun and will contain some of the game's global logic. The cannon itself is parented to this base, creating a hierarchy that will make our later work easier because we won't have to worry about composite movements.

15.2. Controls for the Base and Cannon

We will begin by adding a control for rotation around the vertical axis of the base. This will also rotate the gun and the camera because they are parented to the base.

1. Make sure that the Base object is selected (it should be purple; use the RMB to select it if it is not already selected) and click the Add buttons in the RealtimeButtons for each row of sensors, controllers, and actuators. In every row a new LogicBrick will appear, as shown in Figure 15-2.

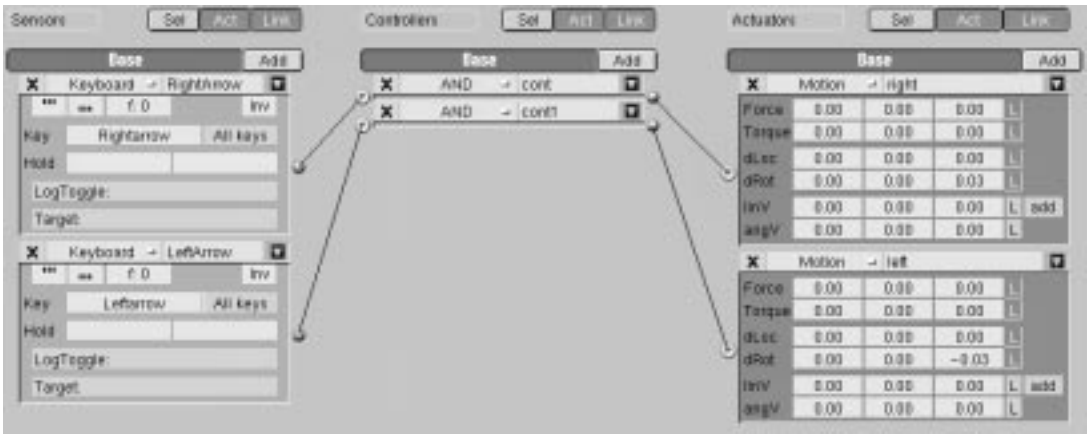


Figure 15-2: LogicBricks to rotate the gun

2. Now link (wire) the LogicBricks by clicking and drawing a line from the little yellow balls (output) to the yellow donuts (input) of the LogicBricks. These connections will pass the information between the LogicBricks.
3. Change the first LogicBrick to a Keyboard Sensor by clicking and holding its MenuButton with the LMB and selecting Keyboard from the pop-up menu.

NOTE *If you have problems with the creating, changing, and linking of LogicBricks, please see the tutorial in Part II.*

4. Now, click the Key field with the LMB and press the RIGHTARROW key when prompted to "Press any key" in the Keyboard Sensor. The Key field should now display "Rightarrow," and the Keyboard Sensor will now react to this key only.
5. Change the third number in the dRot row of the Motion Actuator to **0.03** by using SHIFT-LMB on the number and entering the value with the keyboard. The three fields always denote the three axes (X,Y,Z) of an object; we will rotate around the Z-axis.

6. Move your mouse over the CameraView and press PKEY to start the game. You should now be able to rotate the gun with the RIGHTARROW key.

NOTE *You should always name your LogicBricks and other newly created elements in your scenes (click the default name and enter a new one) to help you to find and understand your game logic later. See the way we've named our LogicBricks in Figure 15-4, for example.*

7. Use the same procedure as above to add LogicBricks to rotate the gun to the left. Use LEFTARROW as the key in the Keyboard Sensor and enter **-0.03** in the third dLoc field of the Motion Actuator.

As you can see, the space in the RealtimeButtons is getting sparse even though we have only six LogicBricks. To clean up your screen area, sse the Icon in the LogicBricks to collapse the LogicBricks to just a title. (Here's another good reason to properly name your LogicBricks.)

15.2.1. Upward Movement

We want Tube Cleaner to have continuous upward movement within the tube. We could copy the effect we used for the rotation of the gun, but there is an alternative that will give us much more control over the movement and that will also allow the player to move to a specific level of the tube. The method we'll employ uses the capabilities of Blender's game engine and its powerful animation system in tandem.

1. Move your mouse over the CameraView and press ALT-A to have Blender play back all the animations defined in the scene. (Press ESC to stop the playback.) Now you've seen the animations, but so far none of these animations is played by the game engine. We have to tell the objects to play the animation. In this way we can interactively control animations with, for example, play, stop, or suspend.
2. Mouse over the wireframe view and press SHIFT-F6. The window will change to an IpoWindow (see Figure 15-3), meant for displaying and editing Blender's animation curves.

The IpoWindow is organized into axes: the horizontal X-axis shows the time in Blender's animation frames, and the vertical Y-axis shows Blender units. The diagonal line (which shows as yellow in the program) is the animation curve for the movement along the Z-axis of the Base object, representing upward movement for our object. Thus, to move the object 10 units up you could move the Ipo cursor (the vertical line just to the right of the Y-axis, shown as green in Blender) with an LMB-click on frame 10. (The CameraView will reflect this movement immediately.)

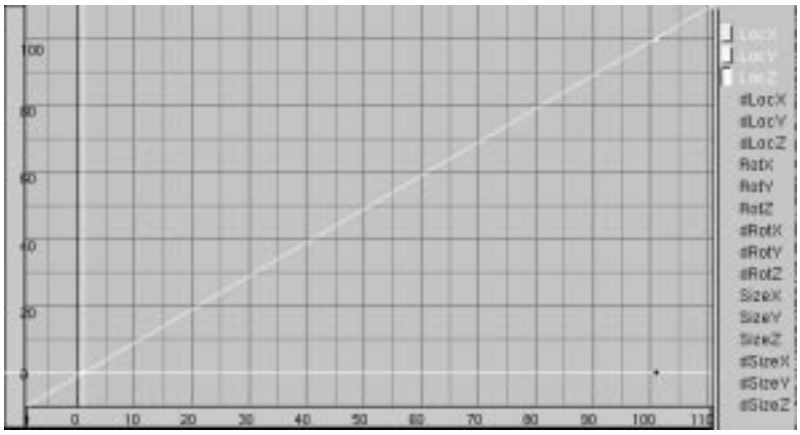


Figure 15-3: IpoWindow with the animation curve representing upward movement

To play this animation in the game engine, we use the "Ipo Actuator" LogicBrick, set to Property type. A *property* is a variable owned by a game object, which can store values or strings.

1. Create a new property to hold the height (zLoc(ation)) of the Base object by clicking "ADD property" in the RealtimeButtons for the Base object.
2. Click on Name and change the default name "prop" to "zLoc" as shown below. This property will hold the height of the gun in the tube.

NOTE *Blender uses capitalization to distinguish between names of objects and properties; "zloc" is not the same as "zLoc."*

Continue creating the LogicBricks shown in Figure 15-4. The Always Sensor triggers the logic for every frame of the game engine animation, ensuring constant movement. The AND Controller simply passes the pulses to two actuators, both of which are connected to the controller and will be activated at the same time.

Figure 15-4: LogicBricks for the upward movement

The Ipo Actuator will play the Ipo animation according to the value in the property zLoc. To produce constant motion we increase the zLoc property every frame with the Property Actuator. Here it is from the type Add, which adds Value (here 0.01) to the zLoc property.

Try different entries in the Value field to get a feel for the speed of the animation. If you'd like to move the cannon downward try entering **-0.01** into the Value field. Once you've finished experimenting a bit enter **0.01** for the value field, as shown in Figure 15-6.

To play the game as it stands so far, move your mouse to the CameraView and press PKEY.

TIP *Blender can show you the Properties in use and their values while the game runs. To do so, choose "Show debug properties" from the Game menu and activate the D button (debug) for every property you'd like to have displayed on screen.*

15.3. Shooting

Let's now add the ability to shoot.

1. Switch the IpoWindow back to a 3DWindow by pressing SHIFT-F5 over the IpoWindow; then select the Gun object with the RMB.

NOTE *You can click every wire from the Gun object but only appropriate selections will be reflected in the ButtonsWindow Header (OB:Gun) and in the RealTime Buttons where Gun will appear in the columns for the LogicBricks.*

2. Now add a sensor, controller, and actuator to the Gun object and wire them as discussed earlier in this tutorial.
3. Change the Sensor to a Keyboard Sensor (name it "Space") and choose (click the Key field) Space as trigger for the gun.
4. Change the Actuator to an Edit Object Actuator. (The default type, Add Object, adds an object dynamically when the actuator is triggered.)
5. Enter **bullet** into the OB field (see Figure 15-5) by clicking it with the LMB and then entering a name from the keyboard. This adds the object "bullet," a premade object, to a hidden layer of the scene.

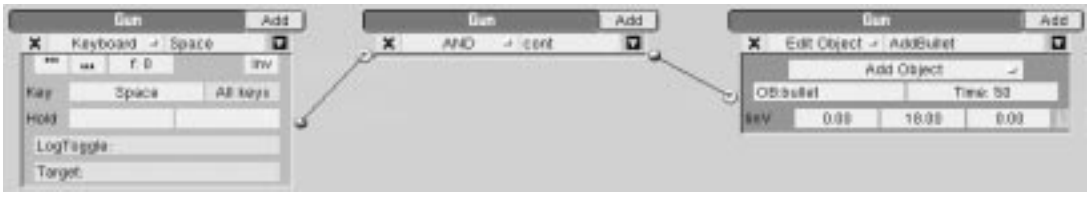


Figure 15-5: LogicBricks to fire the gun

6. Now enter **18.00** (Figure 15-5) as the second number in the linV fields to give the bullet an initial speed.
7. Activate (press) the little L button behind the linV row. This ensures that the bullets will always leave the gun in the direction aimed. Enter **50** in the Time

field to limit the lifetime of the bullets to 50 frames, which will prevent ricochets from bouncing around forever.

Now try to run the game and shoot a bit.

15.3.1. Limiting Our Ammunition

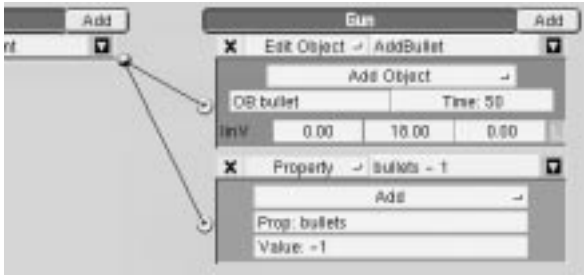
So far we have unlimited ammunition. To change this we again add a property; this time one that stores the number of bullets left.

1. Add a new property by clicking ADD property (as shown in Figure 15-6), then name this property "bullets" and change its type to Int with the MenuButton now labeled Float (the standard type for new properties). (An Int[eger] property only holds whole numbers; this is ideal for our bullets as we don't want half-bullets.) SHIFT-LMB on the field to the right of the Name field and enter **10** to set the number of bullets available at the start of the game to 10.

Figure 15-6: Add property

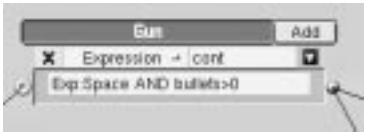
To decrease the number of bullets with every shot we use the same Property Add Actuator that we used to make the base of the cannon move up.

2. Add another actuator by clicking Add in the Gun column of the Actuator, then wire it to the AND Controller we created in the previous step. Change the actuator type to Property and choose Add as the action, then enter **bullets** in the Prop field and **-1** in the Value field. This will subtract 1 (or add -1) from the property "bullets" with every shot triggered by the spacebar.



So far the gun doesn't take any notice of the number of bullets. To change this we use an Expression Controller, which allows us to add single line expressions to process game logic.

3. Change the AND Controller with the MenuButton to an Expression Controller. Then click on the Exp field and enter **Space AND bullets>0** and press ENTER (as shown below). Here "Space" is the name (exactly as you typed it in the LogicBrick) of the Keyboard Sensor and "bullets" is the property. The Controller now only activates the following actuators if the sensor Space is active (meaning that the spacebar is pressed) *and* bullets is bigger than zero.



Try running the game again. You should find that you can only shoot ten times. (Read more about expressions in Section 26.9.)

15.3.2. Making the Bullets Display Function

The last step we need to take to make the gun work is to make the bullets display functional.

Select the display (the right one with the flash on it) with the RMB. It is best to hit the little dot on the 1. The name BulletsDisplay should appear in the RealtimeButtons and in its header as OB:BulletsDisplay. Alternatively, you can zoom into the wireframe view to make the selection easier (see Section 4.10).

Properties

You can see in the RealtimeButtons that there is already a property called Text for the display object. The object has a special text-texture assigned which will display any information in the property called Text that is on it. To test this change the value **10** in the property; the change should be displayed immediately in the CameraView.

Because properties are local to the object that owns them, we have to find a way to pass the value of properties between objects. This is done inside a scene (but will not work across scene borders) with the Property Copy Actuator.

1. Add a line of LogicBricks to the BulletsDisplay like you did before and wire them. Change the Actuator to a Property Actuator, type Copy. See Figure 15-7.

Figure 15-7: LogicBricks to display the number of bullets

2. Enter **Text** into the first Prop field as the name of the property we copy into. Enter **Gun** into the OB field using correct capitalization. (Blender will blank the input field if you enter a nonexistent object.)
3. We will get the value for the number of bullets from the Gun object. Enter **bullets** into field Prop beneath OB as the property name from which we get the value.

   Now start the game again and shoot until you have no more bullets.

15.4. More Control for the Gun

Adding the ability to tilt the gun adds more freedom of movement and makes the game more dynamic. We will use a technique similar to the one we used to create upward movement, by combining animation curves with LogicBricks.

1. Select the gun again, and collapse the LogicBricks by clicking their ▣ arrow icons. (This gives us more space for the logic to come.)
2. The upward movement the gun already contains a motion curve that we can use, but we need to add a property that contains the actual rotation (tilt, rotation around the x-axis). Add a new property by clicking "ADD property" and name it **rotgun**.
3. Use the Add buttons to add a sensor, controller, and one . . . no, two actuators. (You should remember this from the upward movement.) We need one actuator to change the property and one to play the Ipo.
4. Now wire the new LogicBricks as shown in Figure 15-8. The collapsed LogicBricks are the ones you've created for shooting. Change the Bricks as shown in Figure 15-8 and enter all the necessary information.

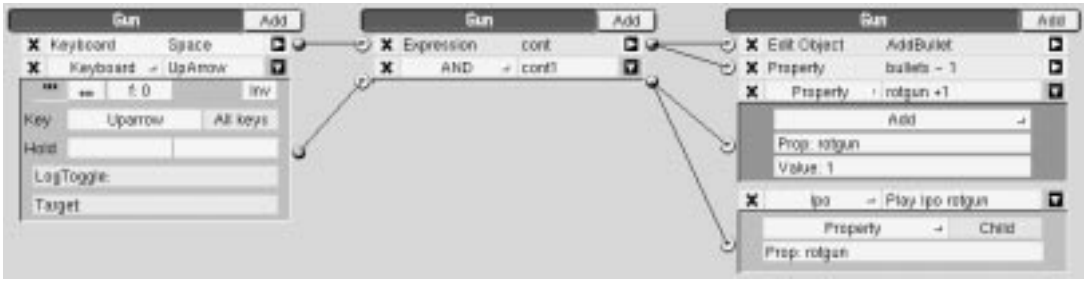


Figure 15-8: LogicBricks to rotate the gun upward

We use the property Add Actuator to increase rotgun by one every time UPARROW is pressed, and we play the Ipo according to the rotgun property to rotate the gun up.

NOTE *I activated the pulse mode ▪▪▪ icon for the Keyboard Sensor to give a keyboard repeat here. This allows us to keep the gun rotating as long as we have the key pressed, without having to release it.*

Now test the rotation. You should see that the gun rotates only a specific distance and then stops.

We control this movement with the animation curve (Ipo). You can see the curve when you switch a window to an IpoWindow with SHIFT-F6 (use SHIFT-F5 to return to the 3DWindow). Note how the curves go horizontally from frame 21 (horizontal axis). This means that no further rotation is possible.

You can also see that we need to make the rotgun negative to rotate it down. Again, add a sensor, controller, and one (yes, this time, it's really only one) actuator. Then wire and name them, as shown in Figure 15-9.

NOTE *We use the Ipo Actuator for the tilting down too. This is perfectly okay, and it allows us to save on creating another LogicBrick. It would also be okay to use a second Ipo Actuator here.*

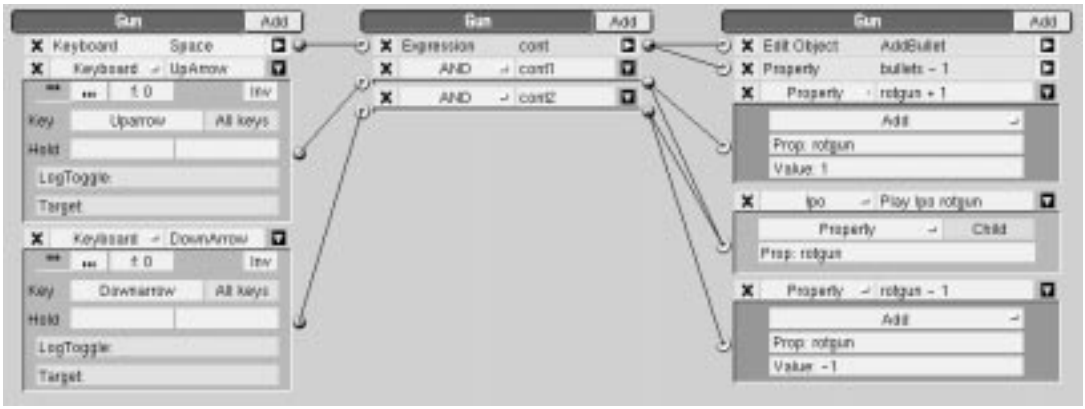


Figure 15-9: Completed LogicBricks allowing us to tilt the gun

TIP *If you prefer "pilot-controls" just swap the Uparrow and Downarrow input in the keyboard sensors.*

There is one drawback to all of this: If you press UPARROW for too long the gun will stop rotating though rotgun is still incremented. This will cause a delay in the gun's rotation back, when you press DOWNARROW again. To correct this we can use expressions again. (See Figure 15-10 for the correct expressions.) These expressions will stop changing rotgun when rotgun is already greater than 21 or less than -21.

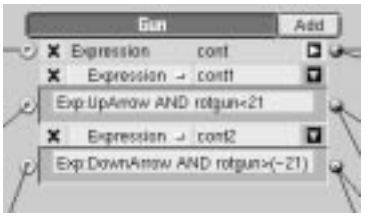
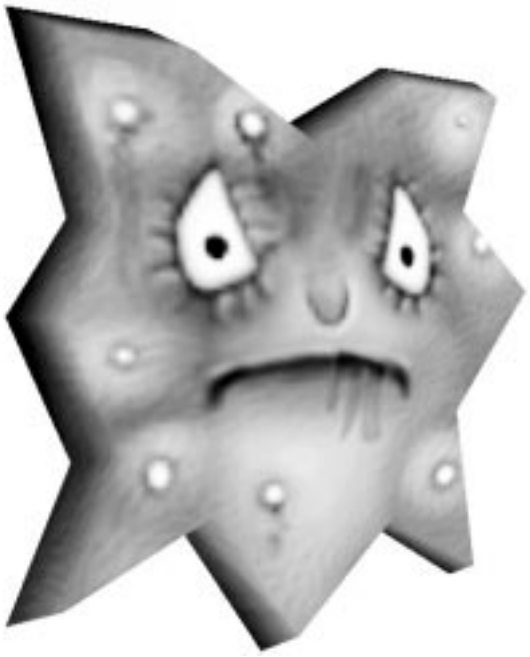


Figure 15-10: Expressions that correctly stop the rotation

CAUTION *It is time to save your project now! Blender scenes are usually very compact, so saving only takes seconds. But first you need to pack (include in the Blender file) the textures, using Pack Data from the ToolsMenu to keep everything together and allow us to send the file to someone. Once your scene is packed, use the FileMenu or save with the keyboard command F2. (See Section 4.3.)*

15.5. Add an Enemy to Shoot



It is now time to add something to shoot at. We want our enemy to:

- React to hits (collisions) with the bullets
- Make a silly face when hit, and die
- Add some points to the player's score

These are all tasks to build into the enemy's game logic.

1. Select the Target object with the RMB. Notice how we've tried to keep the game logic on the target itself? Although we could have added this game logic to the player or any other central element, doing so would make our logic very complex and difficult to maintain. By keeping this game logic local we simplify the logic, make it easier to maintain, and make it much easier to reuse the objects and the logic, even in other scenes.

   We begin again by adding a sensor, controller, and actuator and wiring them. (You should be familiar with this procedure by now.)

   To have our enemy react to a collision, change the Sensor to a Collision Sensor. Enter **bullet** into the Property field to define the name of the property carried by the bullet. In this way the Collision Sensor will only react to collisions with bullets.
2. Change the Actuator to Edit Object and choose Replace Mesh as the type, then enter **TargetDead** into the ME field, as shown in Figure 15-11. This mesh shows the dead target and will be shown from now on when you hit the target. The dead target is on a hidden layer which you can see when you switch layer 11 on and off by pressing SHIFT-ALT-1KEY (see Section 24.1.1).

Figure 15-11: LogicBricks to make the target look silly

15.5.1. Scoring

To score our hit we will use Blender's messaging system, which allows us to send messages from objects to objects. In this case we tell the score display to add some points to our score.

1. Add a second actuator to the target then wire it with the existing controller and change it to a Message Actuator. Leave the To and Body fields blank (as shown below); just fill in the Subject field with **10points**. (This is equivalent to shouting the score into a room at the scorekeeper.)

We now need to set up the score display to react to the score messages.

2. Select the Score-display object and add LogicBricks, as shown in Figure 15-12.

Figure 15-12: LogicBricks to count the score

The Score-display is again an object with a special texture on it showing the content of the Text property (as explained for the bullets display). Be sure to change the 999 to zero or the score will start with 999 points.

The Message Sensor will only hear messages with the 10points subject and then trigger the Property Actuator to add 10 to the Text property, which is then displayed. This makes it very easy to add different scores to different actions simply by adding a new line of LogicBricks to listen for different subjects, then adding the appropriate number of points.

Now try out the game so far and shoot at the enemy. Hits should add 10 points to your score. If anything fails to work, check for correct wiring, and check that the names and capitalization of properties and message subjects are as they should be.

In the final game the targets start to slide down the tube (see Figure 15-13 for a possible solution). The simple target that we've created here also has the drawback that even hitting a dead target will add to your score.

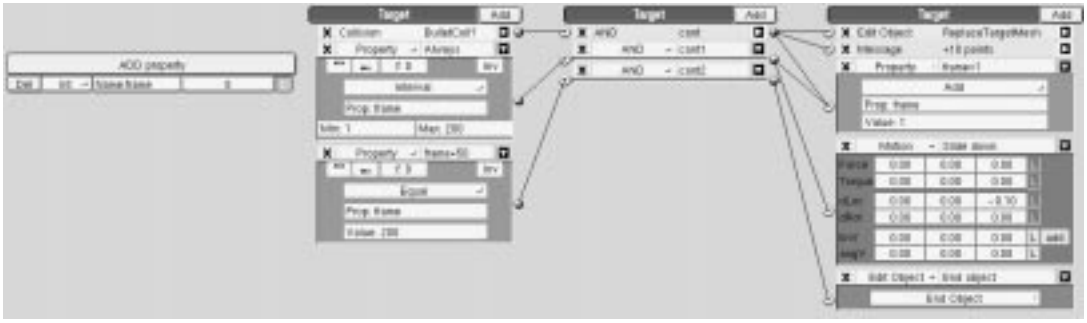


Figure 15-13: Advanced animation for dead targets

We have already used most of the LogicBricks shown in Figure 15-13. Together with the reference (see Chapter 27) and the final game on the CD you can now try to extend the file or just enjoy playing the game.

Whatever you do, don't despair and be sure to keep experimenting. By breaking the task into small steps, you'll be able to tackle even complex logic without getting lost.