Developing web-based curricula using java physlets

Wolfgang Christian and Aaron Titus

Citation: Computers in Physics **12**, 227 (1998); doi: 10.1063/1.168666
View online: https://doi.org/10.1063/1.168666
View Table of Contents: https://aip.scitation.org/toc/cip/12/3
Published by the American Institute of Physics

DEVELOPING WEB-BASED CURRICULA USING JAVA PHYSLETS

Wolfgang Christian and Aaron Titus

Department Editors:
Kevin Zollman
kzollman@phys.ksu.edu
Dean Zollman
dzollman@phys.ksu.edu

HyperText Transfer Protocol (HTTP) on the World Wide Web makes it possible to distribute interactive multimedia-enhanced documents in a platform-independent fashion. HyperText Markup Language (HTML) documents are formulated and transmitted as text documents and can therefore be prepared with any text editor. An HTML text file provides instructions to the browser for locating and displaying multimedia text, graphics, video, and sound files. The recent introduction of the Java programming language by Sun Microsystems now makes it possible to add platform-independent programs to this multimedia stew.[1] Java accomplishes this trick by specifying a relatively simple virtual machine, the Java Virtual Machine (JVM), which can be implemented on any computer architecture including Unix, Mac, and Windows.[2]

Although the JVM does not provide as rich a set of tools as native operating systems, it provides a graphical user interface (GUI) with buttons, a drawing canvas, and other graphical elements. There is virtue in this simplicity. Small platform-independent programs, or "applets," are ideally suited to instructional purposes such as homework problems. These applets can be embedded directly into HTML documents and can interact with the user by means of a scripting language such as JavaScript. This article demonstrates the use of Java applets in conjunction with JavaScript functions to deliver a wide variety of Web-based, interactive physics lessons. Interested readers should consult recent Web Mechanics columns and the many excellent references and tutorials available on the Web for a more complete discussion of HMTL and other Web-based interactive technologies.[3]

Wolfgang Christian is CIP*'s department editor for Book Reviews and professor of physics at Davidson College, Davidson, NC 28036. E-mail: wochristian@davidson.edu*

Aaron Titus is a graduate research assistant in the Department of Physics at North Carolina State University, Raleigh, NC 27695. E-mail: titus@unity.ncsu.edu

Java technology

Preparing Java code to run inside a browser is a multistep process. After writing the code, the programmer compiles it into an intermediate state called a class file. These class files contain the "byte code" for the JVM. Finally, the main class file is embedded into an HTML document using the <APPLET> tag. This embedding is no different from adding a graphic to an HTML page with an <IMG> tag. Class files are downloaded into the browser along with other objects such as sound files or GIF images that are referenced within the containing HTML document. It is up to the browser to lay out the page on the monitor and translate the machine-independent class files into native binary code. It is also the browser's job to provide access to the computer's operating system. Some browsers interpret and execute the Java class files one instruction at a time. The slowness of this process makes Java unsuitable for computationally intensive tasks. Fortunately, browser vendors are now developing compilers that translate the entire class file into native machine code after downloading. These Just-In-Time (JIT) compilers have the potential to make Java almost as fast as C++ code. The current versions of Netscape Navigator and Microsoft Internet Explorer include such compilers.

It is not necessary to become a Java programmer in order to use Java applets in HTML documents. Many Java applets expose their most useful functions and procedures to the outside world—in this case the browser—by declaring them "public," thereby allowing HTML authors to use applets in multiple contexts. The challenge is to find applets that present interesting and pedagogically valuable physics examples.[4] "Physlets" is our name for small, scriptable applets that contain meaningful physics content. In this paper we discuss the design, construction, use, and effectiveness of Physlets.

Embedding

The insertion of tags into the text document specifies the type and location of multimedia content. For example, the <IMG> tag can be used to insert a 300- by 250-pixel

 S0894-1866(98)01403-5

image of an apparatus into a document using the following syntax:

```
<IMG SRC="http://physics.davidson.edu/images/apparatus.gif"
WIDTH="300"HEIGHT="250">
```

Good WYSIWYG HTML editors are akin to the best word processors in hiding such details from the author. The author simply performs routine editing tasks (highlighting, cutting, and pasting) in order to insert images or apply formatting and font styles. The result is a finished document that can be published on the Web. Unfortunately, high-level integration of advanced interactive Web-based technologies such as JavaScript and Java is still sketchy in most authoring packages. A passing acquaintance with HTML syntax is usually required to develop interactive curricular material.

Physlets are ideal for correcting students' misconceptions, since the applets can be scripted to demonstrate nonphysical behavior.

The embedding of applets into Web pages is similar to the embedding of images. The author specifies the name of the applet and its screen size using the <APPLET> tag. For example, the Doppler Physlet shown in Fig. 1 can be embedded within a Web page using the following code:

```
<APPLET CODE="Doppler.class" WIDTH=320 HEIGHT=370>
</APPLET>
```

The applet is displayed by the browser, and the user interacts with the applet by means of its intrinsic controls.[5] The Doppler applet, for example, has a slider to set the velocity and a radio button that enables relativistic effects. In addition, the mouse can be used to make measurements on the wavefronts. The input burden is on the user. The HTML author assumes the user is knowledgeable in the operation of the applet.

A moderately complex applet may be prone to user error and frustration if too many parameters are presented on screen. Most HTML authors prefer to add parameter (<PARAM>) fields to the <APPLET> tag in order to set the default behavior of the applet. Each applet can have a unique set of parameter fields, which the author of the applet should document. Any applet can be embedded on a Web page multiple times with different parameters assigned in order to present different physics problems. The QTime Physlet shown in Fig. 2 has parameter fields for setting the real and imaginary parts of the wavefunction as well as the potential. These fields can be used to set up problems such as particle in a box, simple harmonic oscillator, or barrier penetration.

For example, in order to embed QTime within a Web page so that it shows the evolution in time of a Gaussian wave packet within a square well, we use the following HTML code:

```
<APPLET CODE="QTime.class" WIDTH=320 height=370>
    <PARAM NAME="potential" VALUE="20*(step(1+x)-step(x-
      1))">
    <PARAM NAME="real" VALUE="cos(2*pi*x)*exp(-(x+4)*
      (x+4))">
    <PARAM NAME="imaginary" VALUE="sin(2*pi*x)*exp(-(x+4)*
      (x+4))">
    <PARAM NAME="FPS" VALUE=10>
    <PARAM NAME="dt" VALUE=0.02>
    <PARAM NAME="numPts" VALUE=512>
    <PARAM NAME="minX" VALUE=-10>
    <PARAM NAME="maxX" VALUE=10>
    <PARAM NAME="showControls" VALUE=true>
    <PARAM NAME="helpFile" VALUE="SquareWellHelp.html">
    <PARAM NAME="caption" VALUE="Square Well">
</APPLET>
```

The frames-per-second (FPS) parameter may need to be adjusted to produce smooth animation on less powerful computers. Because Java programmers usually provide default values for parameters, it is often not necessary for the user to assign each and every parameter. Although parameter fields spare the user from having to worry about the

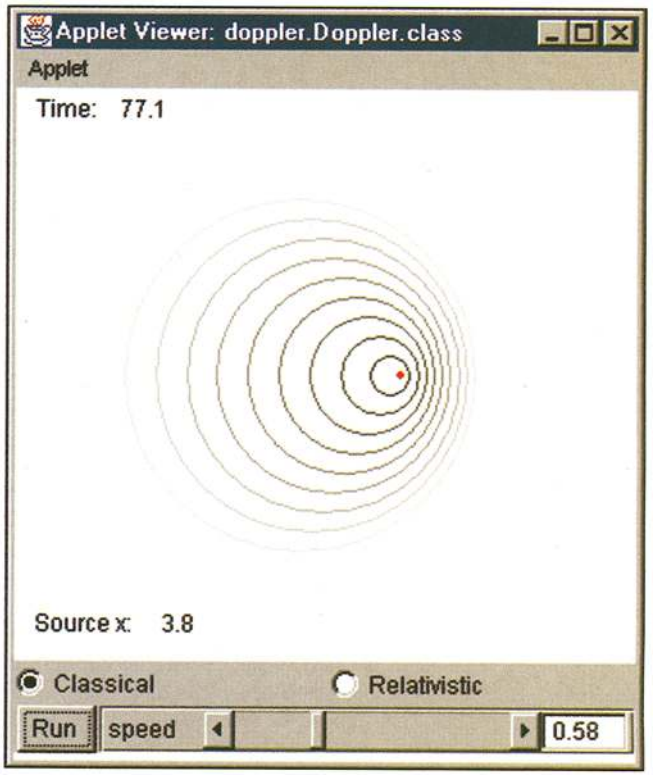

Figure 1. Doppler Physlet displays a slider for setting the source's velocity, which is 0.58 units to the right in this example. The mouse can be used to make measurements on the resulting wavefronts.

arcane details of applets, running an applet is hardly foolproof. The tabbed panel in the QTime interface still allows a user to change the three function strings. This may be desirable for advanced students, but it is unlikely that a sophomore modern-physics student needs to know the details of writing the real and imaginary parts of a Gaussian wavefunction in atomic units with the appropriate momentum boost needed to produce a reasonable group velocity. The showControls parameter is designed to simplify the user interface and to hide these details. Setting this parameter to "false" hides the buttons at the bottom of the applet and the tabbed panel. Starting and stopping the animation or changing the wavefunction must now be done using a scripting language, as explained in the following section.

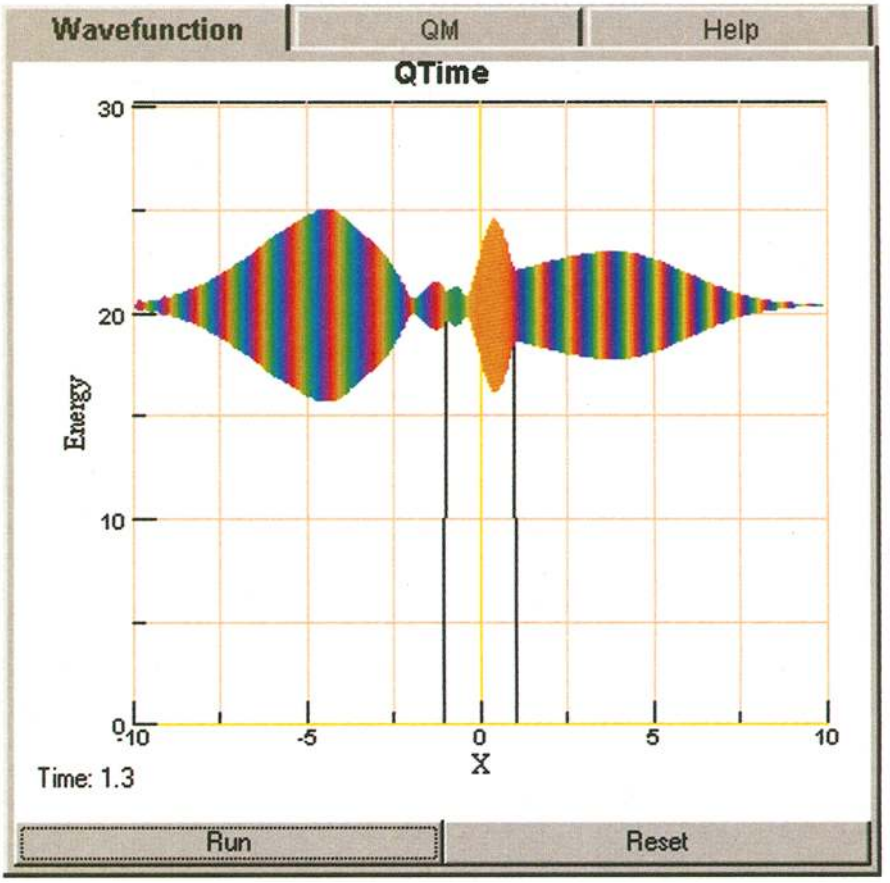

Figure 2. By changing the parameter fields in the embedding script, an HTML author can make the Qtime Physlet represent a particle in a box, a simple harmonic oscillator, or penetration at a barrier. Shown here is a Gaussian wave packet striking a barrier.

Scripting

Scripting makes it possible to change the behavior of an embedded applet after it has been downloaded into a browser, in ways that are just not possible using parameter tags. The user can still interact with the applet but the interaction is controlled by the author, who adds HTML buttons and anchors to produce the desired behavior, using a language such as JavaScript or Visual Basic for Applications.[5] For example, the Animator Physlet is designed to move a geometric shape inside an applet's bounding box along a predefined path, $[x(t), y(t)]$. Creating a 10-pixel-diameter circle that follows a parabolic trajectory requires the following script:

```
document.animator.addCircle(10,"-10+6*t","-5+8*t-4.9t*t")
```

The usual "dot" notation of object-oriented programming can be used to invoke any of the applet's public methods. Methods are procedures or functions on steroids. Unlike traditional subroutines, methods are attached to an object and have access to an object's data structures. In the example code, document refers to the HTML page that contains the applet, animator is the name given to the applet when it is embedded, and addCircle is the name of the method being invoked.

Although animation can certainly be accomplished using more sophisticated programs such as Interactive Physics or possibly even QuickTime movies, Java applets written along the lines of Physlets offer certain advantages. A typical Physlet is less than 100 kbytes long and downloads on an average campus network in a few seconds. Applets download once per session, even if they are embedded on multiple pages. A few lines of script can change the behavior of a Physlet for use in another problem.

Physlets are ideal for correcting students' misconceptions, since the applets can be scripted to demonstrate nonphysical behavior. Standard HTML form fields allow students to change variables and to observe how an applet's behavior changes. It is far more difficult to provide this type of interaction in any type of digital video.

The simplest way to execute a script is to assign it to one of the predefined events of a standard HTML form element. For example, the following code will create two buttons on the HTML page. The first button stops execution of the Physlet animation, whereas the second resets the simulation time to zero.

```
<FORM NAME="Control">
<INPUT TYPE ="button" VALUE ="Stop"
onclick="document.Animator.stop()">
<INPUT TYPE ="button" VALUE ="Reset"
onclick="document.Animator.reset(0.0)">
</FORM>
```

Although this example could be expanded to invoke multiple JavaScript statements within a button's onclick method, more elaborate scripting is best accomplished with JavaScript functions. The following JavaScript function initializes the Ani-

mator applet for one of our interactive homework problems:

```
function prob1(){
      document.Animator.deleteAll();
      document.Animator.reset(0.0);
      document.Animator.setShapeRGB(255,0,0);
      document.Animator.addCircle(20,"0","20-10*t*t");
      document.Animator.setCaption("Constant Acceleration");
      document.Animator.setTimeInterval(0,2);
      document.Animator.forward();
}
```

This script first clears the Animator Physlet of all geometric objects and sets time to zero. A red circle is then created, which moves along the y axis with constant acceleration. Scripts can also be written to create moving rectangles, polygons, arrows, and text, thereby allowing us to create a wide variety of physics problems with just one Physlet. More important, providing students with a visual rather than a textual representation of the information necessary to do a problem affects their problem-solving strategies and opens up the possibility of asking questions that are different from traditional textbook problems. Questions such as "What is the acceleration of the red ball?", "Are the laws of classical dynamics observed in the collision between the red and the blue balls?", and "Which planet in the animation does not obey Kepler's laws?" pose problems in which students must observe a motion and make appropriate measurements to arrive at a solution.

By requiring students to approach problems qualitatively, multimedia-focused problems have a beneficial influence on students' problem-solving and conceptual-reasoning skills.

Although JavaScript functions are part of an HTML page, a browser does not display them. They can most easily be invoked from within the containing page by using a variant of the familiar anchor tag:

```
<A HREF="JavaScript: prob1()">Problem 1</a>
```

The phrase "Problem 1" will be highlighted in blue on the HTML page, but clicking on this anchor will execute the function rather than providing the usual navigation to another page.

Java code

Java is similar to C in some respects, but in philosophy it is much closer to Object Pascal. The following code shows a simple applet that can be embedded and scripted from within an HTML page to display a text message. The paint(Graphics g) method displays a string near the center of the applet. It is invoked whenever the browser determines that the applet's representation on the screen needs to be refreshed. The applet also implements two methods to set and retrieve the displayed message string.

```
import java.applet.*;
import java.awt.*;
public class Hello extends Applet
{
   private String message ="Hello World";
   public void init(){
      this.setBackground(Color.white);
   }
public void setMessage(String m){
   message=m;
    repaint();
 }
 public String getMessage(){
      return message;
 }
  public void paint(Graphics g){ // paint the string
    Rectangle r=this.bounds();
    g.drawString(message, r.width/2, r.height/2);
  }
}
```

The visibility of variables and methods is controlled through the public and private keywords. For example, the variable message is declared private and is not accessible from JavaScript or from other Java classes. Access to this variable is controlled by the accessor methods getMessage() and setMessage(String m). These methods can be invoked from JavaScript without any additional Java code using standard JavaScript syntax:

```
document.hello.setMessage("Another Message")
```

Notice that the setMessage code forces the applet to repaint the screen when the message is changed. The use of methods to access private variables hides the details of the implementation from the JavaScript programmer and is a standard technique of object-oriented programming.

This applet follows the Java Beans specification for the naming of the two accessor methods, even though meeting this specification is not required to create a working applet. The Beans specification does not add any new Java syntax, nor are Beans derived from any Java superclass. Java Beans are standard Java classes that conform to strict naming conventions so that the classes can communicate with other Beans using visual programming and authoring tools. Adding the words set and get to the variable named message allows the authoring package to display a dialog box that

provides quick access to an applet's public methods. The Bean is presented at a very high level of abstraction to the author. A physics instructor will be able to click, drag, and draw a Bean-based Physlet onto a page and obtain a dialog box that displays a documented table of parameters and public methods. In the near future, it will be possible to draw a button and a QMTime Physlet onto an HTML page, drag a connection between these two objects, and connect the button's onclick() method to the Physlet's start() method. The authoring package will create the required JavaScript code the same way it creates the <IMG> tag when an image is dragged into a document.

Java has two important limitations as described in Paul Dubois's recent Scientific Programming article in *CIP*.[6] The first limitation is speed. Although JIT compilers promise considerable speed improvements, it is likely that the added overhead incurred by array checking, garbage collection, and lack of pointer arithmetic will never allow Java to match the speed of well-written Fortran or C code. Sun's recent announcement that the company will differentiate its JIT compilers based upon speed is not a good sign. Only the slow Sun JIT compiler will be free. However, the calculations necessary for most pedagogical applications do not require massive computation. Current Java technology is fast enough to solve and animate the time-dependent Schrödinger equation without flicker using a 120-MHz Pentium processor. But programmers who wish to implement three-dimensional molecular-dynamics simulations in Java will be disappointed.

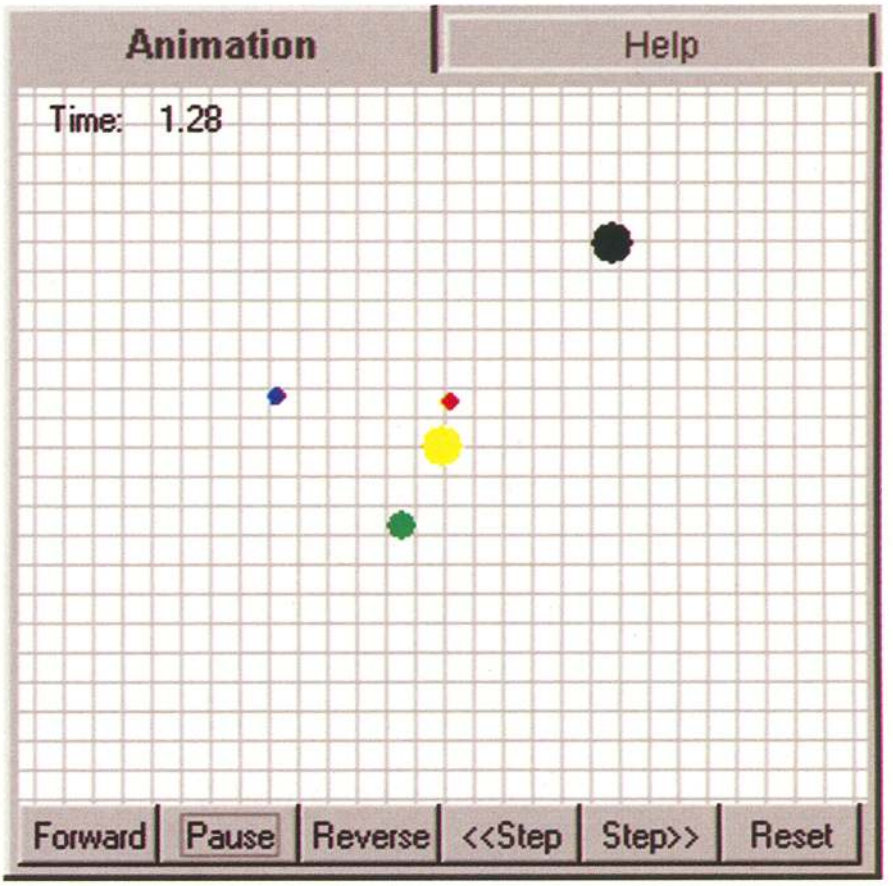

Figure 3. Animator Physlet displays a sun and four orbiting planets, one of which is a pseudo-planet. Through observation and measurement, students discover that the ratio of the square of the orbital period to the cube of the orbital radius is constant in authentic planets.

Java's second limitation is paradoxically its platform independence. In order to maintain both security and platform independence, Applets written in "pure Java" have restricted access to the local operating system. Pure Java applets cannot access Microsoft's ActiveX, Apple's QuickTime, or the local hard disk. Another problem is that the current Abstract Windows Toolkit (AWT) has a limited set of interface components. Modern GUI components such as a tree view or a tabbed panel are simply missing from the AWT. The tabbed panel used in Physlets was constructed from primitive AWT components. Its class file downloads with the applet. Although this approach is feasible, it leads to code bloat and other programming inefficiencies. Both Microsoft and Sun provide extensions to Java to address these problems. Unfortunately, the two companies' solutions are not likely to be compatible. Since platform and vendor independence are important for pedagogical applications, we have attempted to find virtue in simplicity. Physlets are designed for scripting and therefore provide only a minimal interface. Buttons, sliders, and a drawing panel are usually sufficient. Sophisticated interface components are avoided.

The good news about Java is that the core language is well defined and unlikely to change. All major vendors and browser suppliers support the core language. Although speed, access to the local operating system, and user interface remain issues, other language features such as multithreading and network connectivity are built in and a pleasure to use. In short, Java technology is an excellent choice for the development of Web-based curricular material.

Pedagogy

Physlets have been used to create multimedia-focused problems that are fundamentally different from traditional textbook physics problems. In multimedia-focused problems, the information necessary to solve a problem is embedded in the animation rather than given in the text of the question. Students are then required to observe a motion, apply appropriate physics concepts, and make measurements before solving the problem mathematically. Such an approach is remarkably different from the typical strategies of beginning students, who attempt to analyze a problem mathematically before describing it qualitatively. Charac-

terized by a lack of conceptual thought during the problem-solving process, this approach is often referred to by teachers as "plug-and-chug." By requiring students to approach problems qualitatively, multimedia-focused problems have a beneficial influence on students' problem-solving and conceptual-reasoning skills.

Consider the Animator Physlet, for example (Fig. 3). Students are asked to find the "planet" that does not obey Kepler's laws. How do students solve this problem? They must must observe the planets' motion and recognize that the ratios of the square of the orbital period to the cube of the orbital radius must be compared. Numerous parameters must be observed and measured, and it is unlikely that the students' first measurements will yield anything unexpected. However, it turns out that even though the outer orbit appears to be nonphysical (because the planet moves very slowly), the orbit obeys Kepler's laws. The genuinely nonphysical innermost planet moves at too slow a rate, even though the orbit appears to be normal. The visual representation of seemingly abstract formulas can be surprising to students.

In a traditional problem,[7] students are given an orbital parameter for a satellite along with another satellite for comparison; this way of setting the problem suggests a path to the solution. In comparison, the multimedia-focused problem requires observation and conceptual reasoning before quantitative analysis. Furthermore, the idea that the outer planets move slowly in comparison to the inner planets is reinforced by visual observation.

Students often feel that multimedia-focused problems are more like real-world problems than those presented in traditional textbooks. After first encountering multimedia-focused problems, many students comment that they are "like virtual laboratories." As in actual experiments, students must determine what is required to solve a problem before attempting a solution. Likewise, the answer depends both on the method of solution and the experimental error. Instructors can use Physlets to ask questions that are similar to what they would ask of students in actual laboratory situations.

As another example, consider constant acceleration. Students have a great deal of difficulty in distinguishing the direction of motion from the direction of acceleration (or force). In one script, we have created an animation showing a ball that moves upward with a constant downward acceleration. A second script displays a ball that moves downward with a constant downward acceleration. We ask the students to determine the acceleration of these objects. Simply watching the motion of the ball and the time ought to allow students to give an order-of-magnitude estimate of the answer.

Students must not only consider how to solve a problem, but also how to solve it with the least amount of experimental uncertainty. In our experience, students who are well versed in problem solving sometimes have little understanding of experimental error. When the constant-acceleration Physlet was delivered on a homework assignment at North Carolina State University (NCSU), one insightful student remarked that the calculated acceleration was different depending on which equation of motion she used. The difference was the result of experimental error, since the error in the measurement of the time at which the object stopped was greater than the error in the measurement of the position at which the object stopped. The Physlet problem led the student to a greater understanding of experimental error. Unfortunately, some students believe that exercises requiring observation, qualitative reasoning, and measurement should not be part of the lecture course, but should be left for the laboratory. Physlet problems are not, at present, given on course tests or on the Medical Colleges Admission Test.

In introductory courses at Davidson College and NCSU, we are investigating the impact of multimedia-focused problems using Physlets on students' problem-solving skills and understanding of physics concepts. Students have greater difficulty in solving these problems than traditional problems in which the necessary information is given in the text of the question. It is likely that students are more comfortable with a "plug-and-chug" approach than with qualitative reasoning. However, incorporating multimedia-focused problems using Physlets into daily instruction can help to improve students' impressions of these problems and to aid them in acquiring the necessary problem-solving skills. Increased attention to qualitative reasoning may also have an impact on students' understanding of physics concepts.

Our goal as curriculum designers and teachers is not to impress, but to meet the needs of learners. Through the development of multimedia-focused problems using Physlets, we believe that we have found a powerful tool for challenging students' understanding of physics and changing their approach to problem solving.

References

1. See Jon Meyer and Troy Downing, *Java Virtual Machine* (O'Reilly & Associates, Sebastopol, CA, 1997).
2. JavaScript reference can be found on the Web at http://developer.netscape.com/tech/javascript/index.html. See also the Netscape JavaScript Guide at http://developer.netscape.com/library/documentation/communicator/jsguide4/index_dvn.htm.
3. Davidson College Physlets can be found at http://Webphysics.davidson.edu. Many other physics applets are listed on the Gamelan Web site, http://www.developer.com. Be sure to ask the applet author's permission to use his/her work if you are unsure about copyright. It is also a nice gesture to send a thank-you to authors of public-domain applets to let them know that their work is appreciated.
4. The class files for the Doppler Physlet must be placed in the same directory as the HTML file in which the applet is embedded for this example to load properly. Use the optional CODEBASE parameter to specify a different directory from the one containing the referring document. For example, adding CODEBASE="http://myserver.college.edu/Java" specifies that the Java class files needed to run Doppler are located on another server in a subdirectory named Java. Relative directory addressing may also be used with the usual dot-dot notation representing a directory above the current directory.
5. See David Flanagan, *JavaScript: The Definitive Guide*, 2nd ed. (O'Reilly & Associates, Sebastopol, CA, 1997).
6. Paul F. Dubois, "Is Java for Scientific Programming?", Comput. Phys. **11**, 611–617 (1997).
7. See, for example, Douglas Giancoli, *Physics*, 5th ed. (Prentice-Hall, Upper Saddle River, NJ, 1998), p. 142.