

## Game Programming

### Lecture #1 Basic Skills

**Introduction** The Game Industry is a multi-billion dollar industry and still growing. Years ago, the technology forced games to have simple designs. Programs could often be developed by a small team of pure hackers with the major requirement being a good idea. Back then, formal training and education was often looked down on. Now, it is almost essential.

Nowadays, games are so complex they require large teams of programmers, designers, artists, testers, advertisers, and producers to organize and develop them. Games are now as complex as, if not more than, the latest blockbuster film. It is interactive entertainment, pure and simple.

The tools and training needed for game development are enormous, and the complexities warrant academic education beyond a single course. In fact, an entire undergraduate curriculum could be (and in some places is) based on game development.

This lecture note is designed to help students to learn fundamental principles that apply to game programming regardless of the language they use to create the game. These principles include gathering input from users, processing game data, and rendering game objects to the screen. This lecture shows you how to use JavaScript to program a simple game. Anyway, **game programming is a habit, a skill, and an art.**

**Important note** It is the instructor's intention to use **HTML, JavaScript, DHTML, and CSS** to explain the game programming concepts. As an art, game programming is independent of language, meaning you can choose any language you feel comfortable to work with the write games in any platform as long as you understand the concepts of game programming. However, the instructor will also use other programming languages (such as Java) to explain game programming skills whenever it is deemed necessary.

Language is just a tool that facilitates the project you are working on. The instructor chooses the following combination, so the students do not need to install any language, SDKs (system development kit), or IDEs. You can begin writing games simply using applications that come with Windows XP/2000/2003.

Once you learn the concept of programming games, you can apply the skills and knowledge to a platform-sensitive language (or scripting language), such as Visual Basic, C++, C#, Java, and so on. Each language, certainly, has its very own supports of game programming. You just need to be familiar with these language-specific supports to develop games using these platform-sensitive languages, because the concepts you learn in this class are fundamental and universal to all languages.

**What is a game?** A satisfying game can be played over and over again and there are different strategies that lead to success. So what is a (computer) game then? Here is an old-fashion definition:

`"A computer game is a software program in which one or more players make decisions through the control of game objects and resources, in pursuit of a goal."`

Note that the definition does not talk about multimedia supports, such as graphics, or sound effect, or in-game movies. Multimedia obviously plays a important role in making nice, appealing games, so a newer definition is:

`"A computer game is a game composed of a computer-controlled virtual universe with multimedia supports that players may`

interact with in order to achieve a goal or set of goals."

**Object-oriented programming** Most game programming languages adopt the object-oriented model, which is a programming paradigm that uses “objects” to design game programs. The term “object” refers to any identifiable item in a game. For example, in a game that has a ball moving from the left to right, the **ball** is an object.

An object has many **properties** (also known as **attributes**) that determine the appearance of the object, the place the object should be, and so on. For example, a ball has size, color, file name, ID (identification), etc.. Each is an individual property of the ball object, but altogether they decide what and how the ball should be.

An object also has functions, such as start, stop, turn left, hide, show, etc.. Many programming languages provide **methods** (functions that are part of the programming language) for programmers to use. The game programmer’s job is to associate appropriate methods with the object, or “to build the use-defined functions into the objects”.

When the player play games, he/she will have much action, such as clicking a mouse button, pressing a key, etc.. These user actions are known as **events**. Most languages provide **event handlers** for game programmers to respond to the events. For example,

Although object-oriented programming is a complicated program paradigm, you only need to learn two “magic” formulas and some basic concepts. These two formulas have two forms: JavaScript and DHTML forms. They will be used throughout the entire class.

In JavaScript, they are:

```
objectID.propertyName = "value";
```

and

```
objectID.MethodName();
```

For example, the following code block use the <img> HTML tags to load an image file **cat.gif**, which is certainly an object, because you can see it (identify it) on the body area of the browser once the HTML page is loaded.

```
<html><body>

</body></html>
```

The following section assigns an ID (which is m1) to the image file.

```
id="m1"
```

In the following section, **src** is the name of a property (meaning the source). It uses the **propertyName = “value”** format to assign cat.gif as the value of <img> tag.

```
src="cat.gif"
```

The following section uses the entire **objectID.propertyName = "value";** format to tell the computer “when the player click on the image, change the value of the **m1** object’s **src** property to **dog.gif**.”

```
onClick="m1.src='dog.gif' "
```

**JavaScript** JavaScript is the premier client-side scripting language used today on the Web. It’s widely used in

tasks ranging from the validation of form data to the creation of complex user interfaces. Yet the language has capabilities that many of its users have yet to discover.

JavaScript can be used to manipulate the very markup in the documents in which it is contained. As more developers discover its true power, JavaScript is becoming a first class client-side Web technology, ranking alongside (X)HTML, CSS, and XML. As such, it will be a language that any Web designer would be remiss not to master. This chapter serves as a brief introduction to the language and how it is included in Web pages.

JavaScript is not Java language, and does have direct relation to Java language. Java was developed and created by Sun Micro, while JavaScript is solely developed and created by Netscape.

JavaScript code must be embedded into an HTML document using the `<script>` and `</script>` tags. The format:

```
<script language="JavaScript">
JavaScript code fragments
</script>
```

In object oriented languages, object variables are called "properties" and object functions are called "methods." For example,

```
<html><script language=JavaScript>
window.alert("This is a warning message")
</script></html>
```

The `alert()` method of the window (the browser window) object displays a warning message on the Web page.

JavaScript use the following syntax:

- **objectID.method()**: for example, `window.prompt()`. The `prompt` method of the window object creates a pop-up prompt box.
- **objectID.property**: for example, `navigator.appName`. Identify the `appName` property of the navigator object.
- **object.method(object.property)**: for example, `document.write(navigator.appName)`. This displays the value of the `appName` property on the Web page.

When a web page is loaded, the browser's JavaScript interpreter automatically creates objects for most of the components contained in the document. This includes -- but is not limited to -- objects for the document's forms, images, and hyperlinks. According to Netscape, the developers of JavaScript, every page has the following objects:

- **navigator**: has properties for the name and version of the Navigator being used, for the MIME types supported by the client, and for the plug-ins installed on the client.
- **window**: the top-level object; has properties that apply to the entire window. There is also a window object for each "child window" in a frames document.
- **document**: contains properties based on the content of the document, such as title, background color, links, and forms.
- **location**: has properties based on the current URL.
- **history**: contains properties representing URLs the client has previously requested.

Other built-in JavaScript Objects include:

- **Array**: an Array object is used to store a set of values in a single variable name.
- **Date**: The Date object is used to work with dates and times.
- **String**: The String object is used to work with text.
- **Math**: The built-in Math object includes mathematical constants and functions.

For example,

```
<script language="JavaScript">
document.write (document.lastModified);
</script>
```

## Declaring variables

When a car runs, the speed changes all the time. The meter does the reading of speed, and it gives you different values every time when the speed changes. Consider the following table:

time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
speed	65	68	67	66	65	67	68	69	70	71	70	68	67	68	67	66	65	64	66	64

During the last 20 minutes, the value of speed varies up and down. The value is never constant; it varies, so it is a variable.

In the world of game programming, when a programmer needs to handle a property with its value changing all the time, he/she creates (or declare) a “variable”. Variables hold the value when the users enter them. Variables hold the values till the computer notify them to stop holding them.

You can think of variables as "holding places" where values are stored and retrieved. Each holding place has a unique name, which is any combination of letters and numbers that does not begin with a number. In JavaScript, the characters a-z, A-Z, and the underscore character "\_" are all considered to be unique letters.

JavaScript was loosely designed, so it does not require you to state the kind of data type that will be associated with a variable. The syntax JavaScript use to declare a variable is:

var VariableName

The “var” keyword is not strictly required

For example,

```
var mileage = 10987; // number
var CompanyName = "Orange Inc. "; // string
var Readable = "true" // logical / boolean
var score;
var style = null;
```

Any kind of data type can be stored in any variable. Common types of values include:

- Numbers — any number, (ex. 5 or 3.14567)
- Strings or Text — any strings enclosed in double quotation marks, (ex. “\$12.34”, “Hello!”)
- Logical (Boolean) — either true or false
- null — a special value that refers to nothing

Basic rules:

- Case Sensitive Very important... JavaScript is case sensitive, which means, upper case letters and lower case letters are interpreted as being different. For example, the variables “apple” and “Apple”, “ApPle”, “APPLE” are considered four individual variable names.
- Do not use the following words as variable names:

abstract	delete	goto	null	this
boolean	do	if	package	throw
break	double	implements	private	throws
byte	else	import	protected	transient
case	extends	in	public	true
catch	false	instanceof	return	try
char	final	int	short	typeof
class	finally	interface	static	var
const	float	long	super	void
continue	for	native	switch	while
default	function	new	synchronized	with

- Do not use such symbols as 8 ^ \$@+() {}[]\|~!?'"; as part of variable name.

In the following example, “current\_time” is the name of variable that represents a certain registers on the computer memory (DRAMs). This variable holds the readings of data and time, and set the readings ready for the user to use.

With var keyword	Without var keyword
<pre>&lt;html&gt;&lt;script&gt; function st() {   var current_time=new Date()   alert(current_time); } &lt;/script&gt; &lt;button onClick=st()&gt;Click&lt;/button&gt; &lt;/html&gt;</pre>	<pre>&lt;html&gt;&lt;script&gt; function st() {   current_time=new Date()   alert(current_time); } &lt;/script&gt; &lt;button onClick=st()&gt;Click&lt;/button&gt; &lt;/html&gt;</pre>

The user clicks the button to call the st() function, and the st() function in turns force the alert() method to display the most current readings on a warning message box.

Create user-defined functions

When creating games, you need to use computer statements to instruct the computer what to do. Each statement is considered an individual instruction. However, it is very inconvenient to give out instruction one by one, because the computer responds to each instruction one by one.

Now, try another approach. Pack all the statements together into a batch, and then name the batch with a uniquely identifiable ID. You can then execute all the statements packed in that batch by simply calling the ID. In JavaScript, such batch is known as a user-defined function.

A JavaScript function is a collection of instructions (i.e., statements or commands) whose purpose is to accomplish a well-defined task, based on the implementation of existing JavaScript objects, methods, and properties. When the browser executes a function, it is very similar to a VCR executes the pre-programmed action. In JavaScript, a function's input is called its parameters; its output is called the return value. There are two types of functions: built-in functions and custom functions: built-In and custom functions.

In JavaScript, the keyword “function” creates a custom function. A function is defined inside the <SCRIPT> and </SCRIPT> tags; it consists of the keyword function, followed by the following things:

- A name for the function;
- A list of arguments for the function (enclosed in parentheses and separated by commas); and
- JavaScript statements defining the function (enclosed in curly braces { } ).

The syntax:

```
function functionName(arguments) {
  JavaScript statements
}
```

Additionally, once a function has been defined, it must be called in order to execute it. For example, a JavaScript function whose purpose is to compute the square of a number might be defined as follows:

```
<html><script>
function sqr(n) {
  document.write(n*n);
}
```

```

</script>
<input type=text name=t1 onClick=sqr(t1.value)>
</html>

```

In this "sqr" function, sqr(n), the parenthesized variable "n" is called a parameter. It represents the input entered to the text box. The output is equal to number squared. Most of all, the user must click the text box to execute the sqr() function.

## CSS basics

Cascading Style Sheets (CSS) is a way to control the appearance and dynamics of objects in a HTML document. It allows you to convert static HTML objects into motion-enabled objects in an organized and efficient manner.

You can use any existing HTML tag that formats objects in the browser's body area as a "selector" by following the syntax:

```

selector { PropertyName1:value1; PropertyName2:value2.....}

```

For example, the pair of <i> and </i> are HTML tags that italicizes contents in between them. However, it cannot specify color and font size. Compare the following codes and their outputs.

Without CSS	With CSS
<pre> &lt;html&gt;&lt;body&gt; &lt;i&gt;This is a sentence.&lt;/i&gt; &lt;/body&gt;&lt;/html&gt; </pre>	<pre> &lt;html&gt; &lt;head&gt;   &lt;style&gt;     i {color: red; font-size:24px}   &lt;/style&gt; &lt;/head&gt; &lt;body&gt;   &lt;i&gt;This is a sentence.&lt;/i&gt; &lt;/body&gt; &lt;/html&gt; </pre>
<i>This is a sentence.</i>	<i>This is a sentence.</i>

Notice that you must define your CSS property sets in between <style> and </style>. And you are supposed to place your CSS definitions between <head> and </head>. Never place them between <body> and </body>.

Pseudo-classes are also used in CSS to add different effects to some selectors, or to a part of some selectors. When you want one tag but different styles. In a web page you want different type style for different paragraph. Pseudo Class can be defined in embedded style or an external style.

The syntax for setting the style is:

```

.classID {property: value}

```

The syntax for calling the style is:

```

<selector class="classID">...</selector>

```

Consider the following example,

```

<html><style>
.p1 {color:red;font-family:arial}
.p2 {color:green;font-family:times new roman}
</style>
<p class=p1>This is red.</p>
<p class=p2>This is green.</p>
</html>

```

There are two pseudo classes: p1 and p2. They are pseudo because they are not an existing HTML tag (like <p> tag). Now that they are pseudo classes to “p”, they will automatically inherit the functions the <p> tag has—creating a paragraph. The p1 class, however, is assigned two styles:

```
{color:red;font-family:arial}
```

The p2 class is assigned two styles:

```
{color:green;font-family:times new roman}
```

To call the two classes, use

```
<p class=p1>..  
</p>
```

and

```
<p class=p2>..  
</p>.
```

Integrate CSS  
with JavaScript

In most cases, CSS properties are converted to JavaScript properties by removing the data from the CSS property name and capitalizing any words following the dash. For example, the CSS property **background-color** becomes **backgroundColor** in JavaScript.

```
<html>  
<u id=u1 onClick=u1.style.visibility="hidden">Hi!</u><br>  
This is the line 2.  
</html>
```

In addition to identify each style, the keyword “this” is a good alternative.

```
<html>  
<u onClick=this.style.display="none">Hi!</u><br>  
This is the line 2.  
</html>
```

With new Internet Explorer, even the following works!

```
<html>  
<u onClick=style.display="none">Hi!</u><br>  
This is the line 2.  
</html>
```

When click the word “Hi!”, the first line disappears, but the line remains.

DHTML basics

DHTML is Dynamic HyperText Markup Language. As the name conveys, 'dynamic' is something that changes at any given time, but the fact is that, every detail has to be pre-coded. DHTML is a combination of HTML, CSS and JavaScript. DHTML is a feature of Netscape Communicator 4.0 and Microsoft Internet Explorer 4.0 and 5.0 and is entirely a "client-side" technology.

Many features of DHTML can be duplicated with either Flash or Java, DHTML provides an alternative that does not require plug-ins and embeds seamlessly into a web page.

DHTML is useful for creating an advanced functionality web page. It can be used to create animations, games, applications, provide new ways of navigating through web sites, and create out-of-this world layout that simply aren't possible with just HTML.

The underlying technologies of DHTML (HTML, CSS, JavaScript) are standardized, the manner in which Netscape and Microsoft have implemented them differ dramatically. For this reason, writing DHTML pages that work in both browsers can be a very complex issue.

A simply definition of DHTML is: Dynamic HTML is simply HTML that can change even after a page has been loaded into a browser.

DHTML consist of:

- DOM (Document Object Model): Core of DHTML
- CSS: Style provider of DHTML
- Scripting Language: JavaScript (or Jscript) as function provider of DHTML

To sum all this up: CSS and "old HTML" are what you change, the DOM is what makes it changeable, and client-side scripting is what actually changes it. And that's dynamic HTML.

Additionally, Netscape and Internet Explorer use fundamentally different approaches to create DHTML pages. For example, Netscape use the <layer> tag to display a Web page inside a Web page, while Internet Explorer uses <iframe> to do so. It is the intention of this lecture to focus on Internet Explorer.

Consider the following DHTML code example,

```
<html>
<u style="cursor:hand" onClick=alert("Hi!")>Click</u>
</html>
```

This code is a combination of HTML, CSS, and JavaScript. <html> and <u> are generic HTML tags; style="cursor:hand" is a CSS style definition; onClick is a JavaScript event handler and alert("Hi!") is a JavaScript method.

Consider the document.body.clientWidth and document.body.clientHeight properties, and you will understand why DHTML can be used to create games.

DHTML can be used to position objects based on the location and measurement of other objects. The clientWidth and clientHeight properties retrieve the width of the object. The offsetWidth and offsetHeight properties retrieve the width and height of the object relative to the layout or coordinate parent, as specified by the offsetParent property.

Consider the following example,

```
<html><script>
function cc() {
alert(document.body.clientWidth+"x"+document.body.clientHeight)
}
</script>
<button onClick=cc()>Click</button>
</html>
```

When the user click the button, a warning message pops up telling the user width and height of the currently opened window, as shown below:



In the following example, we will force the screen size to change when the user click the button.

```
<html><script>
function shrinkIt() {
```

```

x=document.body.clientWidth-100;
y=document.body.clientHeight-100;
window.resizeTo (x,y)
}
</script>
<button onClick=shrinkIt()>Click</button>
</html>

```

First of all, we declare a variable x and assignment the value document.body.clientWidth-100, which means “detect the current window width and subtract 100 from it.” Second, we declare the y variable and assign the value of y=document.body.clientHeight-100 to it, which means “detect the current window height and subtract 100 from it.” Finally, we use the resizeTo() method to change the size of window.

In the following example, the Web page (document) is the parent object, the overflow area is the offset object. When the user click the button, a warning message of “184:200” appears.

```

<html>
<p id=dt STYLE="overflow:scroll; width:200; height:100">Add a long
paragraph here.....</p>
<button
onclick=alert(dt.clientWidth+": "+dt.offsetWidth)>Click</button>
</html>

```

DHTML is “dynamics”, so you can use innerText, outerText, innerHTML, and outerHTML properties to dynamically inserting texts or codes.

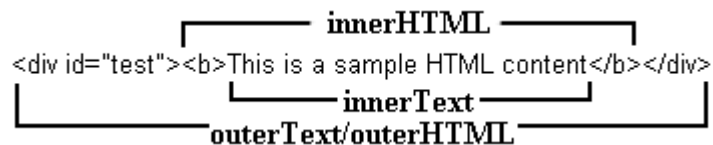
The syntax is:

```

objectID.innerText="strings";
objectID.innerHTML="strings";
objectID.outerText="strings";
objectID.outerHTML="strings";

```

The innerText property sets or retrieves the text between the start and end tags of the object. It is valid for block elements only. By definition, elements that do not have both an opening and closing tag cannot have an innerText property. When the innerText property is set, the given string completely replaces the existing content of the object.



For example,

```

<html>You can order <b id=dt>Pizza</b> here.<br>
<button onclick="dt.innerText='Hamburg'"> Hamburg</button>
</html>

```

Click the “Hamburg” button to replace the word “Pizza” with “Hamburg”.

Consider this example,

```

<P onMouseOver="bt.innerHTML='<b>bold</b>'"
onMouseOut="bt.innerHTML='<i>italic</i>'">
It's <i id=bt>italic</i> now.
</P>

```

The inserted string will be formatted by the HTML tag. Also, test the following code to view the differences.

```
<div id=dt><b>Dynamic HTML!</b></div>
<script>
alert("innerText: "+dt.innerText)
alert("innerHTML: "+dt.innerHTML)
alert("outerText: "+dt.outerText)
alert("outerHTML: "+dt.outerHTML)
</script>
```

However, the output of innerText and outerText look the same.

#### Using JavaScript Functions in DHTML

CSS styles can integrate with JavaScript using DHTML format. One of the beauties of using JavaScript is its capability to pack several commands into one function. It is important to learn how to use JavaScript function in DHTML.

Consider the following example,

```
<html><script language=JavaScript>
function moving() {
dt.style.left="250"; dt.style.top="200";
}
</script>
<p id=dt style="position:absolute;left:25;top:50">This is a
sentence.</p>
<button onClick=moving()>Click</button>
</html>
```

The sentence was originally display at the position (25, 50). When the user click the button, the onClick event handler calls the moving() JavaScript function, which in turn moves the sentence to (250, 200).

Consider another example,

```
<html><script language="JavaScript">
function coloring() {
dt.style.color="blue"; dt.style.backgroundColor="yellow";
}
</script>
<p id=dt style="color:red;background-color:blue"
onClick=coloring()>This is a sentence.</p>
</html>
```

When the user clicks any place on the sentence, the onClick event handler calls the coloring() function, so the colors changes.

Consider the third example,

```
<html>

</html>
```

We can use JavaScript function to add more options of image size.

```
<html>
<script language=JavaScript>
function size1() {img1.style.width="200"}
```

```

function size2() {img1.style.width="300"}
function size3() {img1.style.width="400"}
</script>
<img id=img1 src=http://www.geocities.com/cistcom/bs.jpg
style="width:100">
<button onClick=size1()>Size 1</button>
<button onClick=size2()>Size 2</button>
<button onClick=size3()>Size 3</button>
</html>

```

Game programming is just the matter doing it!

This tutorial should have give n you a rough idea of the things that matter when trying to create a good computer game. But in the end the best way to learn is to do it yourself. Although game programming may sound a big project to you at the current stage, it really is not that far from what you have learned in any programming course. It takes only a new way of thinking, and most of all, a willingness to try. Game programming is just the matter doing it!

There are few simple games created by the instructor as demonstrations. It is time to try them. Do not worry if you don't understand the code. Details will be described in later lectures. For now, just have fun!

Review Questions

1. In a game that has a pitcher who pitches a baseball to a baseball hitter, and a catcher who attempts to catch the baseball. Which is an object?

- A. pitcher
- B. hitter
- C. baseball
- D. all of the above.

2. Given a statement "A red baseball moves from left to right". Which is the property of the baseball?

- A. left to right
- B. red
- C. move
- D. the word "baseball"

3. Given the following code, which is the property of the image file?

```

```

- A. img
- B. src
- C. cat.gif
- D. m1

4. Given the following code, which is the event handler?

```

```

- A. id
- B. src
- C. onClick
- D. img

5. Given the following code, which is the method?

```
window.alert("Welcome!")
```

- A. window
- B. alert()

- C. Welcome!
- D. All of the above

6. Given the following code segment, which is a boolean variable?

```
var selected = "true";  
var Saturday = "Yes";  
var vitamine = "good";  
var monitor = "white";
```

- A. selected
- B. Saturday
- C. vitamine
- D. monitor

7. Given the following code segment, which is a user-defined function?

```
function sqr(n) {  
    document.write(n*n);  
    window.alert(n+1);  
    n++;  
}
```

- A. sqr();
- B. document.write()
- C. window.alert()
- D. n++

8. Given the following CSS code segment, which is a property?

```
<style> p {border: solid 1 black;}</style>
```

- A. <style>
- B. p
- C. border
- D. solid 1 black

9. Given the following CSS code segment, which is a class?

```
<style>  
p1 {border: solid 1 black;}  
p2 {border: solid 1 blue;}  
p3 {border: solid 1 red;}  
.p4 {border: solid 1 blue;}  
</style>
```

- A. p1
- B. p2
- C. p3
- D. .p4

10. Given the following code, which statement is correct?

```
<html><script language="JavaScript">  
function coloring() {  
    dt.style.color="blue"; dt.style.backgroundColor="yellow";  
}  
</script>  
<p id=dt style="color:red;background-color:blue"
```

```
onClick=coloring()>This is a sentence.</p>  
</html>
```

- A. the onClick event handler calls the coloring() function.
- B. when the coloring() function executes, the dt object turns blue.
- C. when the coloring() function executes, the dt object's background color turns yellow.
- D. All of the above

## Game Programming

### Lab #1

### Introduction to Game Programming

#### Preparation #1: Download image files

1. Create a new directory named **C:\lab1**.
2. Use Internet Explorer to go to **<http://business.cypresscollege.edu/~pwu/cis261/download.htm>** to download lab1.zip (a zipped) file. Extract the files to C:\lab1 directory. Make sure the C:\lab1 directory contains:
  - ani\_cat.gif
  - ball.gif
  - bat.gif
  - gopher1.gif
  - gopher2.gif

#### Preparation #2: Sign up a free Web Hosting Service

1. Use Internet Explorer to go to **<http://www.geocities.com>** or **<http://www.tripod.com>** to sign up a free web hosting service. Be sure to learn to upload files and view your uploaded files through the web.

#### Learning Activity #1: A dropping letter

Note: Do not worry if you don't understand the code. Details will be described in later lectures. For now, just have fun!

1. Change to the C:\lab1 directory.
2. Use Notepad to create a new file named **C:\lab1\lab1\_1.htm** with the following contents:

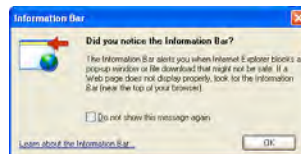
```
<html>

<script>
function init() {
    if (obj.style.pixelTop >= document.body.clientHeight) {
        obj.style.pixelTop = 0; }
    else {obj.style.pixelTop++; }
    s1 = setTimeout("init()", 10);
}
</script>

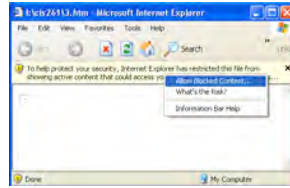
<body onLoad=init()>
<p id="obj" style="position:absolute; top:0">o</p>
</body>

</html>
```

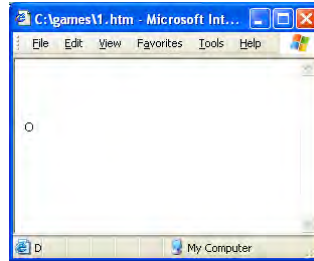
3. Use Internet Explorer (do not use other browser) to execute the file. **Be sure to have your speaker or earphone ready** to hear the sound!
4. If the “Did you notice the Information Bar?” warning box appears, as shown below, click OK.



5. If the default information bar remains, right click the bar, and then select “Allow Blocked Content...”, as shown below.



6. You should now see a letter o dropping from top to bottom over and over again.



## Learning Activity #2: Using animated Gif file as visual effects

Note: Do not worry if you don't understand the code. Details will be described in later lectures. For now, just have fun!

1. Change to the C:\lab1 directory.
2. Use Notepad to create a new file named **C:\lab1\lab1\_2.htm** with the following contents:

```
<html>

<body onLoad=run()>

<script>
var i=0;
```

```
function run() {
```

```
    m1.style.left=i;
    i+=5;
    if (i>=document.body.clientWidth - m1.style.pixelWidth) {i=0;}
```

```
    setTimeout("run()",70);
}
```

```
</script>
```

```

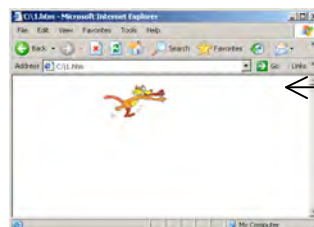
```

```
</body></html>
```

`document.body.clientWidth` represents the current browser width

`m1.style.pixelWidth` is the width of ant\_cat.gif

3. Use Internet Explorer to execute the file. The output now looks:



The cat will run across the screen.

### Learning Activity #3: Effects of distance

Note: Do not worry if you don't understand the code. Details will be described in later lectures. For now, just have fun!

1. Change to the C:\lab1 directory.
2. Use Notepad to create a new file named **C:\lab1\lab1\_3.htm** with the following contents:

```
<html>
<script>
function cc() {
  m1.style.width = m1.style.pixelWidth + 10;

  if (m1.style.pixelWidth < 400) {
    setTimeout("cc()", 50);
  }
  else {dd(); }
}

function dd() {
  m1.style.width = m1.style.pixelWidth - 10;

  if (m1.style.pixelWidth > 10) {
    setTimeout("dd()", 50);
  }
  else {cc(); }
}

</script>
<body onLoad=cc()>
<center></center>
</body>

</html>
```

3. Use Internet Explorer to execute the file. The bat will fly toward you and then backward.



### Learning Activity #4: Your First Game

Note: Do not worry if you don't understand the code. Details will be described in later lectures. For now, just have fun!

1. Change to the C:\lab1 directory.
2. Use Notepad to create a new file named **C:\lab1\lab1\_4.htm** with the following contents:

```
<html>

<script>
function init() {
```

```

ball.style.pixelLeft=Math.floor(Math.random()*250);
ball.style.display="inline";
fall();
}

function fall() {
  if (ball.style.pixelTop>275) {ball.style.pixelTop=0;}
  else { ball.style.pixelTop+=5;}

  if (ball.style.pixelLeft>=375) {ball.style.pixelLeft=0;}
  else { ball.style.pixelLeft+=Math.floor(Math.random()*5);}

  if (ball.style.pixelLeft+15<=bar.style.pixelLeft+50 &&
      ball.style.pixelLeft>=bar.style.pixelLeft &&
      ball.style.pixelTop>=bar.style.pixelTop)
  { clearTimeout(s1); ball.style.display="none";}
  else { s1=setTimeout("fall()",30);}
}

function move() {
var e=event.keyCode;
  switch (e) {
    case 37:
      if(bar.style.pixelLeft <= 0) {bar.style.pixelLeft = 0;}
      else { bar.style.pixelLeft -= 5;}
      break;
    case 39:
      if(bar.style.pixelLeft+50 >= 400) {bar.style.pixelLeft = 350;}
      else { bar.style.pixelLeft += 5;}

      break;
  }
}
</script>

<body onLoad="init()" onKeyDown="move()">

<div style="position:absolute; width:400; height:300; background-color:green;
  border:solid 2 red; top:10; left:10">



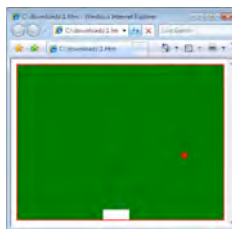
<span id="bar" style="background-color:white; width:50; height:20;
  position:absolute; top:277; left:165; border-bottom:solid 2 red"></span>

</div>

</body>
</html>

```

3. Use Internet Explorer to execute the file. To play the game, use Left (←) and Right (→) arrow keys to move the bar. A sample output looks:



## Learning Activity #5: Random gophers

Note: Do not worry if you don't understand the code. Details will be described in later lectures. For now, just have fun!

1. Change to the C:\lab1 directory.
2. Use Notepad to create a new file named **C:\lab1\lab1\_5.htm** with the following contents:

```
<html>

<style>
img {position:absolute;}
</style>

<script>
function gplace() {
g1.style.pixelLeft = Math.floor(Math.random()*(document.body.clientWidth-50));
g1.style.pixelTop = Math.floor(Math.random()*(document.body.clientHeight-50));

g2.style.pixelLeft = Math.floor(Math.random()*(document.body.clientWidth-50));
g2.style.pixelTop = Math.floor(Math.random()*(document.body.clientHeight-50));

g3.style.pixelLeft = Math.floor(Math.random()*(document.body.clientWidth-50));
g3.style.pixelTop = Math.floor(Math.random()*(document.body.clientHeight-50));

gjump();
}

function gjump() {
var i = Math.floor(Math.random()*3)+1;
switch (i) {
case 1:
g1.src="gopher2.gif";g2.src="gopher1.gif";g3.src="gopher1.gif";
break;

case 2:
g2.src="gopher2.gif";g1.src="gopher1.gif";g3.src="gopher1.gif";
break;

case 3:
g3.src="gopher2.gif";g1.src="gopher1.gif";g2.src="gopher1.gif";
break;
}

setTimeout("gplace()", 800);
}

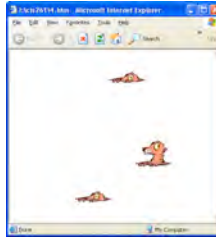
</script>

<body onLoad="gplace()">



</body>
</html>
```

3. Use Internet Explorer to execute the file. Each gopher will stretch its head out of the hole randomly.



### Submittal

Upon completing all the learning activities,

1. Upload the following files to your remote web site (e.g. Geocities.com)
  - ani\_cat.gif
  - ball.gif
  - bat.gif
  - gopher1.gif
  - gopher2.gif
  - lab1\_1.htm
  - lab1\_2.htm
  - lab1\_3.htm
  - lab1\_4.htm
  - lab1\_5.htm
2. Test your program remotely through the web. Make sure they function correctly.
3. Log in to Blackboard, launch Assignment 01, and then scroll down to question 11.
4. Copy and paste the URLs to your games to the textbox, such as:
  - [http://www.geocities.com/cis261/lab1\\_1.htm](http://www.geocities.com/cis261/lab1_1.htm)
  - [http://www.geocities.com/cis261/lab1\\_2.htm](http://www.geocities.com/cis261/lab1_2.htm)
  - [http://www.geocities.com/cis261/lab1\\_3.htm](http://www.geocities.com/cis261/lab1_3.htm)
  - [http://www.geocities.com/cis261/lab1\\_4.htm](http://www.geocities.com/cis261/lab1_4.htm)
  - [http://www.geocities.com/cis261/lab1\\_5.htm](http://www.geocities.com/cis261/lab1_5.htm)

Note: No credit is given to broken link(s).

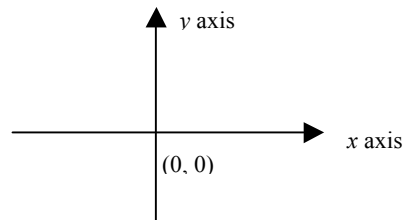
## Game Programming

### Lecture #2 Sprite Programming - Game graphics

**Introduction** The term “graphics” refers to any computer device or program that makes a computer capable of displaying and manipulating pictures. Graphics images are used extensively in games. You are surrounded with graphics as a matter of fact.

Before jumping into the details of how graphics work and how they are applied to games, it's important to establish some ground rules and gain an understanding of how computer graphics work in general. More specifically, you need to have a solid grasp on what a graphics coordinate system is, as well as how color is represented in computer graphics.

**The Graphics Coordinate System** All graphical computing systems use some sort of graphics coordinate system to specify how points are arranged in a window or on the screen. Graphics coordinate systems typically spell out the origin (0,0) of the system, as well as the axes and directions of increasing value for each of the axes. If you're not a big math person, this simply means that a coordinate system describes how to pinpoint any location on the screen as an X-Y value. The traditional mathematical coordinate system familiar to most of us is shown below:



**The Basics of Color** Before this lecture can move on and use some sample code to explain the graphic programming concepts, you need to have some background in computer and web color theory.

The main function of color in a computer system is to accurately reflect the physical nature of color within the confines of a computer, which requires a computer color system to mix colors with accurate, predictable results.

A color monitor has three electron guns: red, green, and blue, and they are called the three color elements. The output from these three guns converges on each pixel on the screen, exciting phosphors to produce the appropriate color. The combined intensities of each gun determine the resulting pixel color.

Technically speaking, Windows colors are represented by the combination of the numeric intensities of the primary colors (red, green, and blue). This color system is known as RGB (Red Green Blue) and is standard across most graphical computer systems.

The following table shows the numeric values for the red, green, and blue components of some basic colors. Notice that the intensities of each color component range from 0 to 255 in value.

Table: Numeric RGB Color Component Values for Commonly Used Colors

Color	Red	Green	Blue
White	255	255	255
Black	0	0	0
Light Gray	192	192	192
Dark Gray	128	128	128
Red	255	0	0

Green	0	255	0
Blue	0	0	255
Yellow	255	255	0
Purple	255	0	255

For example,

```
<html>

<style>
span {width:20; height:70; border:solid 1 black}
</style>

<span id="bar1" style="background-color: red"></span>
<span id="bar2" style="background-color: green"></span>
<span id="bar2" style="background-color: blue"></span>
</html>
```

and the output looks:



Notice that the above colors are represented by using decimal values. In CSS and DHTML, colors can be also defined as a hexadecimal notation for the combination of Red, Green, and Blue color values (RGB). The lowest value that can be given to one light source is 0 (hex #00) and the highest value is 255 (hex #FF). For example,

Color	Color HEX	Color RGB
Black	#000000	rgb(0,0,0)
Red	#FF0000	rgb(255,0,0)
Lime	#00FF00	rgb(0,255,0)
Blue	#0000FF	rgb(0,0,255)
Yellow	#FFFF00	rgb(255,255,0)
Aqua	#00FFFF	rgb(0,255,255)
Purple	#FF00FF	rgb(255,0,255)
White	#FFFFFF	rgb(255,255,255)

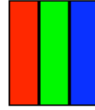
Consider the following code, it uses a both hexadecimal and decimal color values to specify color schemes.

```
<html>

<style>
span {width:20; height:70; border:solid 1 rgb(0,0,0)}
</style>

<span id="bar1" style="background-color: rgb(255,0,0)"></span>
<span id="bar2" style="background-color: #00FF00"></span>
<span id="bar2" style="background-color: #0000FF"></span>
</html>
```

The output looks:



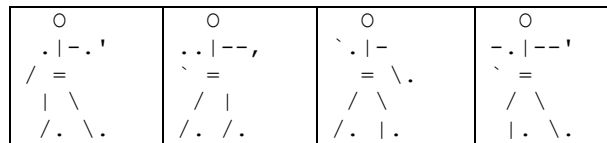
Some years ago, when most computers only supported **256** different colors, a list of **216** Web Safe Colors was suggested as a Web standard. The reason for this was that the Microsoft and Mac operating system used 40 different "reserved" fixed system colors (about 20 each).

We are not sure how important this is now, since most computers today have the ability to display millions of different colors, but the choice is left to you. The 216 cross-browser color palette was created to ensure that all computers would display the colors correctly when running a 256 color palette.

Sprite  
programming

A sprite is a general term for a graphic that can be moved independently around the screen to produce animated effects. Many sprites are programmed by the game programmer and are created after the game starts, so they are also known as movable object blocks. This type of sprites is actually a code blocks that is temporarily stored in memory and is transferred to the screen.

Given the following text arts (also known as ASCII arts), each is an individual character-formed graphic simulating a stage of a human's dancing motion.



By displaying one of them at a time in a sequence and then repeat the sequence over and over again, you can create a sprite of dancing man. In the following code, each ASCII art is placed in between <pre> and </pre>. The HTML <pre> tag defines preformatted text. The text enclosed in the pre element usually preserves spaces and line breaks. The text renders in a fixed-pitch font.

```
<body onLoad=init()>
<pre id="h1" style="display:none">
  O
  .|-.'
 /  =
 |  \
/.  \.
</pre>

<pre id="h2" style="display:none">
  O
  ..|--'
  =
 /  |
/.  /.
</pre>

<pre id="h3" style="display:none">
  O
  `.-|
  = \.
 /  \
/.  |.
</pre>

<pre id="h4" style="display:none">
  O
```

```

- . | -- '
\  =
 /  \
| .  \ .
</pre>

</body>

```

Each `<pre>` tag is assigned an ID--h1, h2, h3, h4, so they can be easily identified. They are also configured by **display:none**, so they will not be displayed by default.

In order to dynamically control the sprite, you can add the following JavaScript code with DHTML support.

```

<script>
var i=1;
function init() {
if (i>4) { i=1; }
else {
clear();
switch (i) {
case 1: h1.style.display='inline'; break;
case 2: h2.style.display='inline'; break;
case 3: h3.style.display='inline'; break;
case 4: h4.style.display='inline'; break;
}
i++;
}
s1 = setTimeout("init()", 100);
}

function clear() {
h1.style.display='none';
h2.style.display='none';
h3.style.display='none';
h4.style.display='none';
}
</script>

```

The **init()** function uses `i++` to continuously adding 1 to the value of *i*. The following line forces the computer to set the maximum to 4. Literally it means “if the value of *i* is greater than 4, the value of *i* must return to 1”.

```

if (i>4) { i=1; }

```

The following code block forces the computer to apply codes based on the value of *i*. For example, when *i* = 3, on the codes for case 3 are executed, the rest are ignored.

```

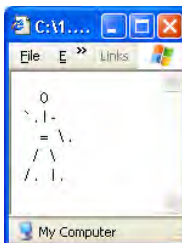
switch (i) {
case 1: h1.style.display='inline'; break;
case 2: h2.style.display='inline'; break;
case 3: h3.style.display='inline'; break;
case 4: h4.style.display='inline'; break;
}

```

The **setTimeout()** method forces the computer to execute the `init()` function every 100 milliseconds.

The **clear()** function sets each ASCII art to be hidden again (`display='none'`). This function clears the background, so you will not see double-, triple-, or multiple-images.

A sample output looks:



Some sprites are pre-created animated .gif files. This kind of sprites is rectangular graphic created by combining multiple GIF images in one file. In other words, they are a batch of images, displayed one after another to give the appearance of movement.

Although graphics design is not a topic of this course, Appendix A provides a short lecture on how you can create animated GIF files.

Create  
graphics  
programmatically

New game programmers usually relate this “term” to a pre-designed software-produced graphics, but tend to forget the fact that many computer languages are also capable of producing graphics programmatically.

Consider the following code, it uses CSS (cascade style sheet) to specify how an area defined by `<span>...</span>` in a code division (defined by `<div>...</div>`) should look in a Web browser's body area.

```
<html>
<style>
.wKey {
  background-color:white;
  border-top:solid 1 #cdcdcd;
  border-left:solid 1 #cdcdcd;
  border-bottom:solid 4 #cdcdcd;
  border-right:solid 1 black;
  height:150px;
  width:40px
}
</style>

<div>
<!-- white keys -->
<span class="wKey" id="midC"></span>
</div>

</html>
```

→ embedded CSS style sheet

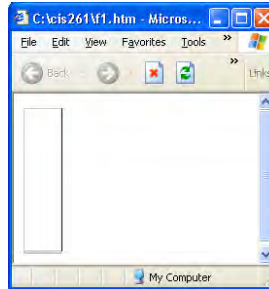
CSS uses `<style>...</style>` to embed a style sheet onto an HTML code. In this example, `wKey` is the name of style, which contains the following attributes and their associated values in the format of *attributeName : value*.

- **background-color : white.** This set of attribute and value simply defines the background of the object using this style as *white*.
- **border-top : solid 1 #cdcdcd.** This sets the top border to be a solid line with size of 1px and color value of #cdcdcd (in the RGB format). This color combination produces the color of light gray.
- **height : 150px.** This sets the height of the object to be 150px.
- **width : 40px.** This sets the height of the object to be 40px.

The following line associates an object with ID **midC** with the **wKey** style, such that all the styles wKey has will apply to this midC object.

```
<span class="wKey" id="midC"></span>
```

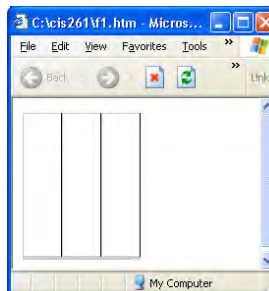
The output of this code looks:



In order to add two more objects that are clones of the midC object, simply add the following two lines. Just be sure to assign new IDs for each of the two new objects!

```
<span class="wKey" id="midC" style="left:50"></span>
<span class="wKey" id="midD" style="left:91"></span>
<span class="wKey" id="midF" style="left:132"></span>
```

The output now looks:



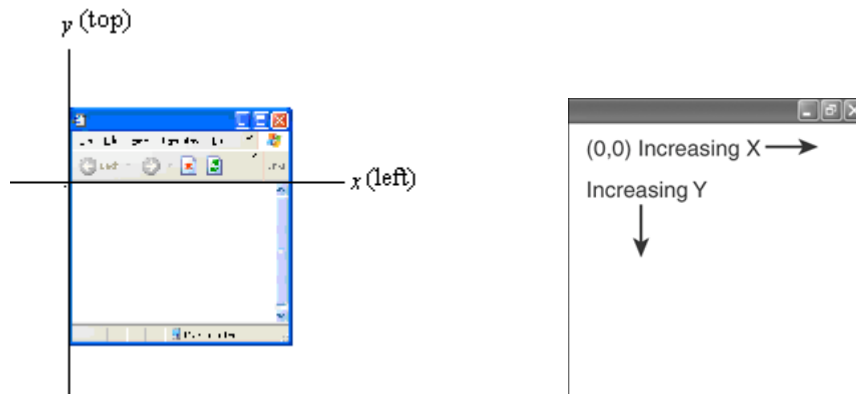
Take a close look at these three objects, they are placed closely together, there's no space between any consecutive ones. In order to dynamically place each object on the web browser's body area, you need to use absolute positioning system. You do so by adding the following bold line:

```
.wKey {
    position: absolute;
    background-color: white;
    border-top: solid 1 #cdcdcd;
    border-left: solid 1 #cdcdcd;
    border-bottom: solid 4 #cdcdcd;
    border-right: solid 1 black;
    height: 150px;
    width: 40px
}
```

DHTML, with CSS, use two pre-defined variables -- *left* and *top* -- to represent *x*- and *y*-coordinate of the browser's body area. In fact, most graphics in a Windows program are drawn to the client area of a window, which uses the Windows graphics coordinate system.

The following figures explains how the coordinates of the client area begin in the upper-left

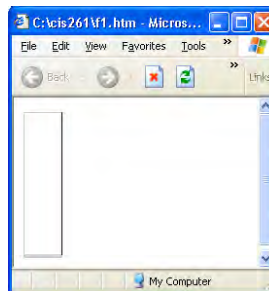
corner of the window and increase down and to the right, as you learned earlier in the hour. This coordinate system is very important because most GDI graphics operations are based on them.



Consequently, by adding the following bold line, all the three objects' y-coordinate are set to be 20px, which means they all are 20px below the x-axis.

```
.wKey {
  position:absolute;
  top:20;
  background-color:white;
  border-top:solid 1 #cdcdcd;
  border-left:solid 1 #cdcdcd;
  border-bottom:solid 4 #cdcdcd;
  border-right:solid 1 black;
  height:150px;
  width:40px
}
```

The output now looks weird. There are 3 objects, supposedly, but only one is displayed. What's wrong?



The answer is, the absolute positioning system uses both *left* and *top* to specify a given object of absolute position. All the three objects' *top* values are defined as 20px in the above code segment. It's time to assign each object's *left* value individually.

```
<span class="wKey" id="midC" style="left:50"></span>
<span class="wKey" id="midD" style="left:91"></span>
<span class="wKey" id="midE" style="left:132"></span>
```

The above uses a technique known as **inline style**, which means to add a style to the HTML tag as an attribute of the tag. The syntax is:

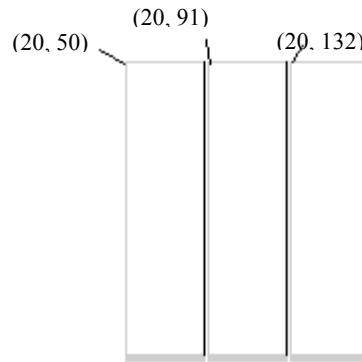
```
style = "attributeName1:Value1;attributeName2:Value2;...;
        attributeName:Valuen"
```

The midC object is now assigned an initial point (20, 50), midD (20, 91), and midE (20, 132). The distance between 50 and 91 is 41. In this code example, 41 is determined because each object is 40px wide (as specified by width:40px), plus 1px as the space between two objects.

$$50 + 40 + 1 = 91$$

$$90 + 40 + 1 = 132$$

Take a good look at the output. There is a visible space between every two objects.



To make the above graphics looks like a section of piano keyboard, add the following bold lines:

```
<html>
<style>
.wKey {
  position:absolute;
  top:20;
  background-color:white;
  border-top:solid 1 #cdcdcd;
  border-left:solid 1 #cdcdcd;
  border-bottom:solid 4 #cdcdcd;
  border-right:solid 1 black;
  height:150px;
  width:40px
}

.bKey {
  position:absolute;
  top:20;
  background-color:black;
  border:solid 1 white;
  height:70px;
  width:36px
}
</style>

<div>

<!-- white keys -->

<span class="wKey" id="midC" style="left:50"></span>
<span class="wKey" id="midD" style="left:91"></span>
<span class="wKey" id="midE" style="left:132"></span>

<!-- black keys -->
<span class="bKey" id="cSharp" style="left:72"></span>
```

➔ A new CSS style

```

<span class="bKey" id="dSharp" style="left:114"></span>

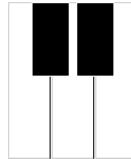
</div>

</html>

```

The first added code segment defines a new CSS style named **bKey**, which specifies how a black key should look. The second simply adds two black keys in the division ( as defined by <div> ... </div>). Each black key has a unique ID.

The output now looks (Surprisingly, you create a graphic using DHTML codes!):



Creating  
animated  
graphics  
programmatically

To add animated effects to the above graphics, you need to write **user-defined functions**. Such functions can perform visual effects. For example, you can add visual effects to the keys, so they will have an effect of key-down-key-up.

Consider the following added bold codes:

```

<span class="wKey" id="midC" style="left:50" onMouseOver="CDown () "  

onMouseOut="CUp () "></span>

```

It uses two **event handlers**: *onMouseOver* and *onMouseOut* to call two different user-defined functions: *CDown()* and *Cup()*. Such programming techniques will be discussed in a later lecture. For now, you just have to know what they are.

Two functions are created by adding the following lines to the HTML file:

```

<script>
<!-- for middle C -->
function CDown() {
midC.style.borderTop="solid 1 black";
midC.style.borderLeft="solid 1 black";
midC.style.borderBottom="solid 1 black";
midC.style.borderRight="solid 1 #cdcdcd";
}

function CUp() {
midC.style.borderTop="solid 1 #cdcdcd";
midC.style.borderLeft="solid 1 #cdcdcd";
midC.style.borderBottom="solid 4 #cdcdcd";
midC.style.borderRight="solid 1 black";
}
</script>

```

These two user-defined functions, *CDown()* and *Cup()*, are combinations of DHTML statements. The syntax is (notice that **style** is a keyword):

```

objectID.style.AttributeName = "Value";

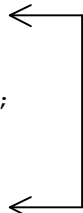
```

In *CDown()*, the first line changes the top border of the **midC** object to “solid 1 black”, the second changes the left border, the third changes the bottom border, and the fourth changes the right border to “solid 1 #cdcdcd”.

Compare CDown() and CUp() carefully and you will find out their color values in the reversed order to each other. Also, be sure to know that the color values of CUp() are exactly the same as those defined in wKey style.

```
function CDown() {
  midC.style.borderTop="solid 1 black";
  midC.style.borderLeft="solid 1 black";
  midC.style.borderBottom="solid 1 black";
  midC.style.borderRight="solid 1 #cdcdcd";
}

function CUp() {
  midC.style.borderTop="solid 1 #cdcdcd";
  midC.style.borderLeft="solid 1 #cdcdcd";
  midC.style.borderBottom="solid 4 #cdcdcd";
  midC.style.borderRight="solid 1 black";
}
```



By executing these two functions, the color of midC's top border changes from #cdcdcd to black, and then back to #cdcdcd. This creates an effect of key-down-key-up. A complete code is available in the learning activity, please learning this technique for your own good.

The advantage of creating graphic programmatically is that you can easily apply dynamic functions to the graphics.

Adding visual effects to graphics programmatically

A technique called **alpha** is greatly used today to create visual. Alpha is a value representing a pixel's degree of transparency. The more transparent a pixel, the less it hides the background against which the image is presented. In PNG, alpha is really the degree of opacity: zero alpha represents a completely transparent pixel, maximum alpha represents a completely opaque pixel. But most people refer to alpha as providing transparency information, not opacity information.

Many other techniques are available as explained in the following table, which contains commonly used DHTML effects. The syntax of using filter is:

```
filter:effectName(parameter)
```

Table: Common filter effects:

Effects	Description	Example
alpha()	Set a transparency level	{filter:alpha(opacity=2)}
blur()	Creates the impression of moving at high speed	{filter:blur(strength=5)}
chroma()	Makes a specific color transparent	{filter:chroma(color=#008855)}
dropshadow()	Creates an offset solid silhouette	{filter:dropshadow(color=#0066cc)}
fliph	Creates a horizontal mirror image	{filter:fliph()}
flipv()	Creates a vertical mirror image	{filter:flipv()}
glow()	Adds radiance around the outside edges of the object	{filter:glow(color=#ccffdd)}
grayscale()	Drops color information from the image	
invert()	Reverses the hue, saturation, and brightness values	{filter:invert()}
light()	Project light sources onto an object	
mask()	Creates a transparent mask from an object	
shadow()	Creates a solid silhouette of the object	{filter:shadow(direction=10)}
wave()	Create a sine wave distortion along	

	the x- and y-axis	
xray()	Shows just the edges of the object	{filter:xray()}

For example, you can use some of these functions to create visual effects of texts.

```
<html><head><style>
font {font-family:arial;font-size:24px;width:100%;}
</style></head><body>
<font style="filter:glow()">Glow</font>
<font style="filter:blur()">Blur</font>
<font style="filter:fliph()">Flip Horontally</font>
<font style="filter:flipv()">Flip Vertically</font>
<font style="filter:shadow()">Shadow</font>
<font style="filter:dropshadow()">Drop Shadow</font>
</body></html>
```

The output looks:



On the other hand, some of them apply to images and graphics, too. Consider the following example, it demonstrates how the original images are displayed using some of these functions:

```
<html>
<style>
img {width:100}
</style>

<table>
<tr><td>Original</td><td>Alpha:</td><td>Gray:</td>
<td>Invert:</td><td>Xray:</td></tr>
<tr><td>

</td><td>

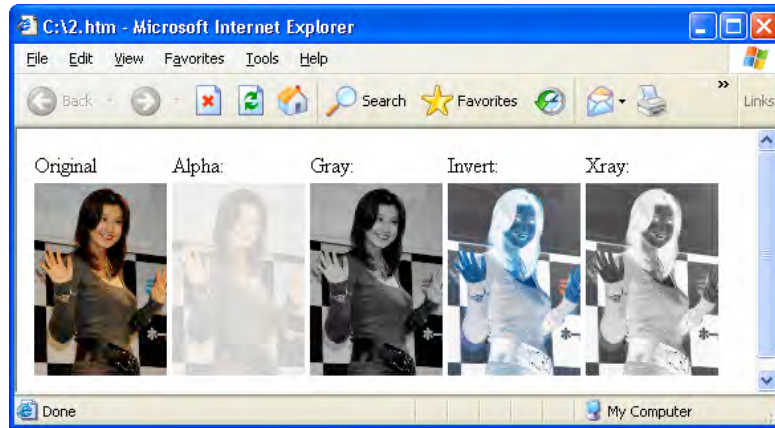
</td><td>

</td><td>

</td><td>

</td></tr></table>
</html>
```

The output looks:



## Review Questions

1. Which is the color combination of "red"?

- A. #0000FF
- B. rgb(255,255,0)
- C. #00FF00
- D. rgb(255,0,0)

2. Given the following code segment, what background color will the object have?

```
style="background-color: #00FF00"
```

- A. red
- B. green
- C. blue
- D. yellow

3. Given the following code segment, which is the closest name of the color?

```
border-top:solid 1 #cdcdcd;
```

- A. lime
- B. grey
- C. orange
- D. pink

4. Given the following code segment, which statement is correct?

```
<script>
img { width:35; height:40}
</script>
```

- A. the image height is set to be 40 centimeters
- B. the image width is set to be 35 pixels
- C. the image height is set to be 40 dot-per-inch
- D. the image width is set to be 35 inches

5. Given the following code segment, which is the correct coordinate combination of the object "dot"?

```
dot.style.right=50;
dot.style.left=200;
dot.style.top=150;
dot.style.bottom=100;
```

- A. (50, 200)
- B. (50, 150)
- C. (200, 150)
- D. (150, 100)

6. Which is the correct way to create a glow effect?

- A. `<span style="filter:glow()">Glow</span>`
- B. `<span style="glow()">Glow</span>`
- C. `<span style="filter:glow">Glow</span>`
- D. `<span style="filter(glow)">Glow</span>`

7. \_\_\_ is a value representing a pixel's degree of transparency.

- A. Alpha
- B. chroma
- C. fliph
- D. shadow

8. A(n) \_\_\_ is a general term for a graphic that can be moved independently around the screen to produce animated effects.

- A. image
- B. graphic
- C. sprite
- D. gif

9. Given the following code, which statement is incorrect?

```
<pre id="h1" style="display:none">
  O
  .|-.'
 /  =
 |  \
/.  \.
</pre>
```

- A. It is an object, and its name is h1.
- B. It is defaulted to be displayed as the word "none".
- C. using `h1.style.display='inline'` can make it visible.
- D. It is defaulted to be invisible.

10. Given the following code segment, which statement is correct?

```
function init() {
    .....
    i++;
    .....
}
```

- A. It uses `i++` to continuously adding 1 to the value of i.
- B. It uses `i++` to continuously adding + to the value of i.
- C. It uses `i++` to continuously adding ++ to the value of i.
- D. It uses `i++` to continuously adding i to the value of i.

## Game Programming

### Lab #2

### Game Graphics

#### Preparation #1:

1. Create a new directory named **C:\games**.
2. Use Internet Explorer to go to **<http://business.cypresscollege.edu/~pwu/cis261/download.htm>** to download lab2.zip (a zipped) file. Extract the files to C:\games directory.

#### Learning Activity #1: Basic Sprite Programming I

1. Change to the C:\games directory.
2. Use Notepad to create a new file named **C:\games\lab2\_1.htm** with the following contents:

```
<html>
<script>
var i=1;
function init() {
if (i>4) { i=1; }
else {
clear();
switch (i) {
case 1: h1.style.display='inline'; break;
case 2: h2.style.display='inline'; break;
case 3: h3.style.display='inline'; break;
case 4: h4.style.display='inline'; break;
}
i++;
}
s1 = setTimeout("init()", 100);
}

function clear() {
h1.style.display='none';
h2.style.display='none';
h3.style.display='none';
h4.style.display='none';
}
</script>

<body onLoad=init()>
<pre id="h1" style="display:none">
  O
  .|-.'
 / =
 | \
/. \.
</pre>

<pre id="h2" style="display:none">
  O
  ..|--,
  \ =
  / |
 /. /.
</pre>

<pre id="h3" style="display:none">
  O
  \.|-
  = \.

```

```

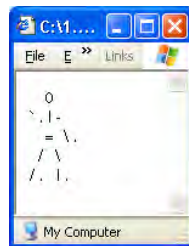
/ \
/. |.
</pre>

<pre id="h4" style="display:none">
  0
  .|--'
  \ =
  / \
  |. \.
</pre>

</body>
</html>

```

3. Use Internet Explorer to execute the file. A sample output looks:



## Learning Activity #2: Basic Sprite Programming II

Note: Do not worry if you don't understand the code. Details will be described in later lectures. For now, just have fun!

1. Change to the C:\games directory.
2. Use Notepad to create a new file named **C:\games\lab2\_2.htm** with the following contents:

```

<html>
<style>
.pm {position: absolute;left=0; top:10 }
</style>

<script>
function init() {

if (ball.style.pixelLeft > 60) { pacman.style.pixelLeft++; }
if (pacman.style.pixelLeft > 70) { ghost.style.pixelLeft++;}

ball.style.pixelLeft++;
s1=setTimeout("init()", 10);
}
</script>

<body onLoad=init()>

<pre id="ball" class="pm">

:~:
|_|
</pre>

<pre id="pacman" class="pm">
  --.
/ o.-'

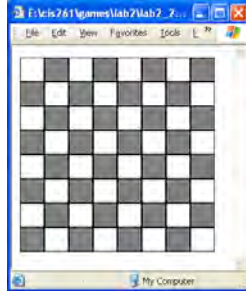
```

```
\ ' - .
' _ _ '
</pre>
```

```
<html>

<style>
.cell { width:30; height:30; border:solid 1 black; }
</style>

<script>
function init() {
code1="";
  for (i=1; i<=72; i++) {
    if (i%9==0) { code1+="
```



#### Learning Activity #4: Creating animated graphics programmatically

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab2\_4.htm with the following contents:

```
<html>
<style>
.wKey {
    position:absolute;
    top:20;
    background-color:white;
    border-top:solid 1 #cdcdcd;
    border-left:solid 1 #cdcdcd;
    border-bottom:solid 4 #cdcdcd;
    border-right:solid 1 black;
    height:150px;
    width:40px
}

.bKey {
    position:absolute;
    top:20;
    background-color:black;
    border:solid 1 white;
    height:70px;
    width:36px
}
</style>

<script>
function Down() {
var keyID = event.srcElement.id;
document.getElementById(keyID).style.borderTop="solid 1 black";
document.getElementById(keyID).style.borderLeft="solid 1 black";
document.getElementById(keyID).style.borderBottom="solid 1 black";
document.getElementById(keyID).style.borderRight="solid 1 #cdcdcd";
}

function Up() {
var keyID = event.srcElement.id;
document.getElementById(keyID).style.borderTop="solid 1 #cdcdcd";
document.getElementById(keyID).style.borderLeft="solid 1 #cdcdcd";
document.getElementById(keyID).style.borderBottom="solid 4 #cdcdcd";
document.getElementById(keyID).style.borderRight="solid 1 black";
}
</script>

<div>

<span class="wKey" id="midC" style="left:50"
    onClick="Down()" onMouseOut="Up()" "></span>
```

```

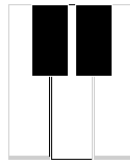
<span class="wKey" id="midD" style="left:91"
onClick="Down()" onMouseOut="Up()" "></span>

<span class="wKey" id="midE" style="left:132"
onClick="Down()" onMouseOut="Up()" "></span>

<span class="bKey" id="cSharp" style="left:72"></span>
<span class="bKey" id="dSharp" style="left:114"></span>
</div>
</html>

```

3. Use Internet Explorer to execute the file. Use the mouse cursor to move among the three keys, you should the visual effects.



### Learning Activity #5: Creating animation programmatically

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab2\_5.htm with the following contents:

```

<html>
<script>
function rotate1() {
m1.src='star2.gif';
m2.src='star1.gif';
m3.src='star2.gif';
m4.src='star1.gif';
m5.src='star2.gif';
setInterval("rotate2()",100);
}

function rotate2() {
m1.src='star1.gif';
m2.src='star2.gif';
m3.src='star1.gif';
m4.src='star2.gif';
m5.src='star1.gif';
setInterval("rotate1()",100);
}
</script>

<body onKeyDown='rotate1() '>





</body>
</html>

```

3. Use Internet Explorer to execute the file. Press any key, so the stars rotate. A sample output looks:



### Submittal

Upon completing all the learning activities,

1. Upload all files you created in this lab to your remote web server.
2. Log in to Blackboard, launch Assignment 02, and then scroll down to question 11.
3. Copy and paste the URLs to the textbox. For example,
  - [http://www.geocities.com/cis261/lab2\\_1.gif](http://www.geocities.com/cis261/lab2_1.gif)
  - [http://www.geocities.com/cis261/lab2\\_2.gif](http://www.geocities.com/cis261/lab2_2.gif)
  - [http://www.geocities.com/cis261/lab2\\_3.htm](http://www.geocities.com/cis261/lab2_3.htm)
  - [http://www.geocities.com/cis261/lab2\\_4.htm](http://www.geocities.com/cis261/lab2_4.htm)
  - [http://www.geocities.com/cis261/lab2\\_5.htm](http://www.geocities.com/cis261/lab2_5.htm)

No credit is given to broken link(s).

## Game Programming

### Lecture #3 Control of flows

**Objective** Game programs are not linear--each statement executes in order, from top to bottom--especially when the players have options. To create interesting games, you need to write programs that execute (or skip) sections of code based on some condition. This lecture discusses about how you can control flows of a game program.

**Introduction** In computer science, control flow (or alternatively, flow of control) refers to the order in which the individual statements or instructions of an imperative program are performed or executed. Within a programming language, a control flow statement is an instruction that when executed can cause a change in the subsequent control flow to differ from the natural sequential order in which the instructions are listed. Discussion of control flow is almost always restricted to a single thread of execution, as it depends upon a definite sequence in which instructions are executed one at a time.

Consider the following example. The numbers at the beginning of each indicates the line of the code. However, the sequence of the execution, which is how the computer will read the statements are different.

```
Li  Code
ne
01  <html><body onLoad=cc() >
02
03  <script>
04  var i=0;
05
06  function cc() {
07  m1.style.left=i;
08  i+=5;
09
10  if (i >= document.body.clientWidth)
11    {i=0;}
12
13    setTimeout("cc()",70);
14  }
15  </script>
16
17  
19
20  </body></html>
```

The first line, `<html><body onLoad=cc() >`, contains an event handler **onLoad** which calls a user-defined function **cc()** to make decision about what the computer should respond to the changing of the *x*-coordinate of an image file that has an ID **m1**. In other word, the execution sequence is not necessary same as the line sequence.

Expressions,  
Statements,  
and Blocks

An **expression** is a construct made up of variables, operators, and method invocations, which are constructed according to the syntax of the language, which evaluates to a single value. You've already seen examples of expressions, illustrated in bold below:

- `var i = Math.floor(Math.random()*10);`
- `p1.innerHTML = "<img src='my.gif'>";`
- `i++;`
- `for (i=0; i<=10; i++)`

**Statements** are roughly equivalent to sentences in natural languages. A statement forms a complete unit of execution. The following types of expressions can be made into a statement by terminating the expression with a semicolon (;).

- Assignment expressions
- Any use of ++ or --
- Method invocations
- Object creation expressions

Such statements are called **expression statements**. Here are some examples of expression statements.

```
x = 8933.234;           // assignment statement
i++;                   // increment statement
document.write("Hello World!"); // method invocation statement
Bicycle myBike = new Bicycle(); // object creation statement
```

In addition to expression statements, there are two other kinds of statements: declaration statements and control flow statements. A declaration statement declares a variable. You have not seen any examples of declaration statements, because JavaScript does not require type declaration. The following is an example used by other languages:

```
double aValue = 8933.234; //declaration statement
```

Finally, **control flow** statements regulate the order in which statements get executed. Details of control flow statement will be discussed in later section.

A **block** is a group of zero or more statements between balanced braces and can be used anywhere a single statement is allowed.

```
for (i=1; i<=300; i++) {
    x = Math.round(Math.random()*screen.width);
    y = Math.round(Math.random()*screen.height);

    code1 = "<span id=dot"+i;
    code1 += " style='left:" + x + "; top:" + y +"'>.</span>";
    document.write(code1)
}
```

Noticeably, operators may be used in building expressions, which compute values; expressions are the core components of statements; statements may be grouped into blocks.

The if-then statement

The **if-then** statement is the most basic of all the control flow statements. It tells your program to execute a certain section of code only if a particular test evaluates to true. In JavaScript, the syntax:

```
if (condition)
{
    code to be executed if condition is true
}
```

Refer back to the above code, an *if.then* statement is used in the format of JavaScript to determine if the value of variable *i* is larger or equal to player's web browser width (which is represented by **document.body.clientWidth**).

```
if (i>=document.body.clientWidth-20)
```

If the evaluation result is true, then the computer executes:

```
{i=0;}
```

If this test evaluates to false (meaning that the value of *i* is less than that of `document.body.clientWidth`), control jumps to the end of the if-then statement and continue with the following statement:

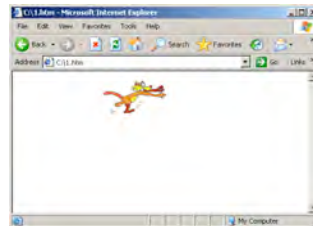
```
setTimeout("cc()", 70);
```

**setTimeout** is a method that evaluates an expression after a specified number of milliseconds has elapsed. The syntax is:

```
setTimeout(vCode, iMilliSeconds [, sLanguage])
```

The above statement uses the `setTimeout` method to evaluate the `cc()` function after 70 milliseconds has elapsed. And, since it is place within the `cc()` function, this `setTimeout` method forces the computer to execute `cc()` every 70 milliseconds. In other words, over and over again.

Obviously, the first line `<body onLoad=cc()>` is only executed once during the entire program life cycle. It's mission is to start the `cc()` function. All the control flow arrangement in `cc()` will determine what the computer should do afterwards. The output thus looks:



The **if..then** statement used in this code provides a simple **decisive logic**, which means “if the condition does not make, forget the whole thing! You don’t get to have the second option!” Refer back to the following code block,

```
if (i >= document.body.clientWidth)
{i=0;}
```

The condition is the part, `i >= document.body.clientWidth`, which can be either true or false. This statement read “if *i* is greater than or equal to the value of the document body’s width of the client’s browser, then *i* equals one”. However, it does not describe what to do if the above condition is false.

The  
if..then..else  
statement

The **if-then-else** statement provides a secondary path of execution when an "if" clause evaluates to false. In JavaScript, the syntax:

```
if (condition)
{
  code to be executed if condition is true
}
else
{
  code to be executed if condition is not true
}
```

In the following example, there are two user defined functions **cc()** and **dd()**. They both use the if..then..else statement to take some action if the expected condition does not happen. To be more specific, **m1** is the ID of an object (which is the bat.gif file) and its value of width increments by 10px every time when the `cc()` is executed (and decrements by 10px when `dd()` is executed).

The condition of `cc()` is that the value of width must be less than 400px. If the condition is false (meaning `m1`'s width value is  $\geq 400$ ), then call the `dd()` function (or you can say to skip `cc()` and move to `dd()` function).

Reversely the condition of `dd()` is to be greater than 10px. If the condition is false (meaning `m1`'s width value  $\leq 10$ ), then call the `cc()` function.

```
<html>
<script>
function cc() {
    m1.style.width = m1.style.pixelWidth + 10;

    if (m1.style.pixelWidth < 400) {
        setTimeout("cc()", 50);
    }
    else {dd(); }
}

function dd() {
    m1.style.width = m1.style.pixelWidth - 10;

    if (m1.style.pixelWidth > 10) {
        setTimeout("dd()", 50);
    }
    else {cc(); }
}

</script>
<body onLoad=cc()>
<center></center>
</body>

</html>
```

DHTML use the following syntax to represent attribute of a given object (Note: style is a keyword).

`objectID.style.AttributeName`

The **pixelWidth** attribute of `m1` is represented by `m1.style.pixelWidth`, and its value is a string by default, such as **20px**. The value `pixelWidth` holds is an integer, so it can be evaluated by comparison operators (e.g.  $>$ ,  $<$ ,  $\geq$ ,  $\leq$ ). For example,

```
if (m1.style.pixelWidth < 400) {
    setTimeout("cc()", 50);
}
else {dd(); }
```

If the condition is true, the computer executes `setTimeout("cc()", 50)` after 50 milliseconds; otherwise, it calls the `dd()` function.

The *if..then..else* statement is good for anything that falls to the two-state phenomena--either true or false. However, most games are not that black or white; they are very colorful, and require more than one simple decisive logics to make decisions. In this case, you can consider using nested *if..then..else* statement, or using the *switch..case* statement.

The  
switch..case  
statement

The **switch..case** statement is an alternative to the If...Then...Else statement. It also makes code easier to read especially when testing lots of different conditions. It also makes the code more efficient. In JavaScript, the **switch..case** statement uses the following syntax:

```

switch (testexpression) {
  Case expressionlist1:
    Statements;
  Case expressionlist2:
    Statements;
  .....
  .....
  default : statement;
}

```

where *testexpression* is a single expression, which is evaluated at the top of the list. The *expressionlist* then lists possible conditions and if the *testexpression* is equal to this case, the code in that section is run.

When using a *switch..case* statement, you need to declare a variable that will be matching against with the options in the case list.

Inside the *switch..Case* statement, you define separate *Case* statements for each condition to be checked against. In this example, you have 5 and each one is set to respond to a different blood type. If a match can be found, computer executes the code immediately following the relevant Case statement.

Another point of interest is the **break** statement after each case. Each **break** statement terminates the enclosing switch statement. Control flow continues with the first statement following the switch block. The **break** statements are necessary because without them, case statements fall through; that is, without an explicit break, control will flow sequentially through subsequent case statements.

The **default** section handles all values that aren't explicitly handled by one of the case sections. It is sometime good idea to have the **default** section, but it depends on the situation.

The random gopher game uses the *switch..case* statement to controls the flow.

```

.....
<script>
.....

switch (i) {
case 1:
  g1.src="gopher2.gif";g2.src="gopher1.gif";g3.src="gopher1.gif";
  break;

case 2:
  g2.src="gopher2.gif";g1.src="gopher1.gif";g3.src="gopher1.gif";
  break;

case 3:
  g3.src="gopher2.gif";g1.src="gopher1.gif";g2.src="gopher1.gif";
  break;
}
.....

</script>
.....

```

The **gjump()** function contains a variable *i*, whose value is assigned by a random number generator -- the **Math.random()** function -- of JavaScript. By default, this random function returns a random number between zero and 1 with the decimal point. The following is a list of possibly results of Math.random():

```
0.47145134906738295
0.8141135558627748
0.696928080402613
0.9209730732076619
0.8254258690882129
0.03148481558914484
```

The results are unpredictable, and they are all smaller than 1. To generate a smaller range of random number in JavaScript, simply use the following code:

```
var variableName=Math.floor(Math.random()*11)
```

where 11 dictates that the random number will fall between 0-10. To increase the range to, say, 100, simply change 11 to 101 instead.

You may be curious as to why **Math.round()**, instead of **Math.floor()**, is used in the above code. While both successfully round off its containing parameter to an integer within the designated range, **Math.floor** does so more “evenly”, so the resulting integer isn’t lopsided towards either end of the number spectrum. In other words, a more random number is returned using **Math.floor()**.

This code uses the following line to randomly generate four possible values 0, 1, 2, and 3.

```
var i = Math.round(Math.random()*3);
```

Since 0 is not a valid case (only 1, 2, and 3 are listed used in the above code), the following excludes the possibility of 0 as a result.

```
if (i < 1) { i = Math.round(Math.random()*3); }
```

Consider the following code block. If the value of *i* returns out to be 1, the computer will only carry out the code of the case 1. The keyword **break** forces the flow to stop, so the codes that belong to case 2 and case 3 will not be read and executed by the computer.

```
switch (i) {
  case 1:
    g1.src="gopher2.gif";g2.src="gopher1.gif";g3.src="gopher1.gif";
    break;
```

Similarly, if the value of *i* returns out to be 2, the computer will only carry out the code of the case 2. Case 1 and case 3 are ignored.

```
case 1:
  g1.src="gopher2.gif";g2.src="gopher1.gif";g3.src="gopher1.gif";
  break;

case 2:
  g2.src="gopher2.gif";g1.src="gopher1.gif";g3.src="gopher1.gif";
  break;

case 3:
  g3.src="gopher2.gif";g1.src="gopher1.gif";g2.src="gopher1.gif";
  break;
```

Technically, the final **break** is not required because flow would fall out of the switch statement anyway. However, we recommend using a **break** so that modifying the code is easier and less error-prone.

The *for* loop      The *for* statement provides a compact way to iterate over a range of values. Programmers often

refer to it as the "for loop" because of the way in which it repeatedly loops until a particular condition is satisfied. The general form of the for statement can be expressed as follows:

```
for (initialization; termination; increment) {
    statement(s);
}
```

When using this version of the for statement, keep in mind that:

- The *initialization* expression initializes the loop; it's executed once, as the loop begins.
- When the *termination* expression evaluates to false, the loop terminates.
- The *increment* expression is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment or decrement a value.

Given the following code:

```
<html>
<style>
span {position:absolute; color:red}
</style>

<script>
code1 = "";
function draw_star() {
    for (i=1; i<=300; i++) {
        x = Math.round(Math.random()*screen.width);
        y = Math.round(Math.random()*screen.height);

        code1 += "<span id=dot"+i;
        code1 += " style='left:" + x + "; top:" + y + "'>.</span>";
    }

    bd.innerHTML = code1;
}
</script>

<body id="bd" onLoad="draw_star();" >
</body>
</html>
```

The *for* loop is set to repetitively add an objects to the browser's body area based on the following logic:

```
<span id=dotn style='left:x; top:y'>.</span>
```

where *n* represents a list of consecutive numbers starting with 1 and end at 300, because the for loop definition:

```
for (i=1; i<=300; i++)
```

*x* and *y* are random numbers generated by the **Math.random()** function multiplied by the screen width and screen height. The **Math.round()** function is used to round up/down the random values to integers.

Since *x* and *y* are inside the *{* and *}* of the *for* loop, the computer will generate a set of random numbers each time when *i* increments (*i++*). There will be 300 sets of randomly generated *x*- and *y*- coordinates, each set is used to place one dot (.) one the browser's body area. In other words, the final client side HTML output consist 300 lines of:

```
<span id=dot1 style='left:x1; top:y1'>.</span>
<span id=dot2 style='left:x2; top:y2'>.</span>
```

```

.....
.....
<span id=dot299 style='left:x299; top:y299'>.</span>
<span id=dot300 style='left:x300; top:y300'>.</span>

```

Obviously the *for* loop is used to shorten the game code. For example, the **cc()** function in the following code uses the *for* loop to repeats itself 13 times. Each time it produce a new HTML statement, which will be inserted to a blank space on the web browser's body area with an ID **areal** (specified by the **<span id=areal></span>** statement).

```

<html>
<SCRIPT>

function cc(){

    for (i=1; i<=13; i++) {
        areal.innerHTML += "<img src=c" + i + ".gif id=c" + i + ">";
    }

}
</SCRIPT>
<body onLoad=cc()>
<span id=areal></span>
</body><html>

```

The above bold code block will produce the following HTML codes, which are to be inserted inside the **<span id=areal></span>** code block.

```

<img src=c1.gif id=c1>
<img src=c2.gif id=c2>
<img src=c3.gif id=c3>
<img src=c4.gif id=c4>
<img src=c5.gif id=c5>
<img src=c6.gif id=c6>
<img src=c7.gif id=c7>
<img src=c8.gif id=c8>
<img src=c9.gif id=c9>
<img src=c10.gif id=c10>
<img src=c11.gif id=c11>
<img src=c12.gif id=c12>
<img src=c13.gif id=c13>

```

The *for* loop is very helpful in producing similar code block repetitively, especially when you need to reproduce a large amount of them. The property **innerHTML**, in DHTML, sets or retrieves the HTML contents between the start and end tags of the object (as in the above example **<span>** and **</span>**). It is valid for block elements only. The syntax is:

```
objectID.innerHTML="HTML contents"
```

Notice that the above code use a **+=** operator, which mean “appending” or “adding without replacing the current content”. This operator is used because when the **innerHTML** property is set, the given HTML string completely replaces the existing content of the object. You can remove the **+** sign (only the enlarged one) from the following statement, and test the code again to see for yourself.

```
areal.innerHTML + "<img src=c" + i + ".gif id=c" + i + ">";
```

Instead of using the **innerHTML** property, you can also consider using the **insertAdjacentHTML** method, as shown in the following code.

```

<html>

<script>
function cc() {
for (i=1; i<=3; i++) {
p1.insertAdjacentHTML("BeforeBegin","<img src='gopher'+i+'.gif'
id=g"+i+"><br>")
}
}
</script>

<body onLoad=cc()>

<span id=p1></span>

</body>
</html>

```

The insertAdjacentHTML method inserts the given HTML text into the element at the location. The syntax is:

```
objectID.insertAdjacentHTML(location, string)
```

Available position values are:

- beforeBegin - Inserts the text immediately before the element.
- afterBegin - Inserts the text after the start of the element but before all other content in the element.
- beforeEnd - Inserts the text immediately before the end of the element but after all other content in the element.
- afterEnd - Inserts the text immediately after the end of the element.

The *for..in* statement

The for..in loop is a bit different from the other loops we've seen so far. It allows you to loop through the properties of a JavaScript object.

If you are unfamiliar with objects in JavaScript, think of them as black boxes that can have a number of properties associated with them. For example, a cat object might have a color property with a value of "black", and an ears property with a value of 2! The basic for..in construct looks like this:

```

for ( variable_name in object_name )
{
    < do stuff with the property here >
    < the current property name is stored in variable_name >
}

```

Note that, on each pass through the loop, the loop variable variable\_name holds the name of the current property. To obtain the value of the current property, you would use the following syntax:

```
object_name[variable_name]
```

For example, this code loops through all the properties of the navigator object (a built-in JavaScript object that holds information about your browser, adds each property's name and value to a string, then displays the resulting string in an alert box:

```

function display_nav_props ( )
{
    var i;
    var output_string = "";

```

```

for ( i in navigator )
{
    output_string += "The value of " + i +
        " is: " + navigator[i] + "\n";
}

alert ( output_string );
}

```

The *while* loop     The **while** loop is used when you want the loop to execute and continue executing while the specified condition is true. The syntax in JavaScript is:

```

while (condition)
{
    code to be executed
}

```

The *while* statement evaluates *expression*, which must return a boolean value. If the expression evaluates to true, the *while* statement executes the *statement(s)* in the *while* block. The *while* statement continues testing the expression and executing its block until the expression evaluates to false.

For example, you can apply a while loop to a shooting game with the logic “while the x-coordinate is greater than or equal to 400, set the x-coordinate value to 95”.

```

<html>

<script>
var i = 95; // declare a variable i
function shoot() {
    b01.style.left=i; //let the current x-coordinate equal i value
    i+=10; // i increment by 10

    while (i >= 400) {
        b01.style.display='none';
        i=95;
    }
    setTimeout("shoot()",20); // call shoot() after 20 milliseconds
}

</script>

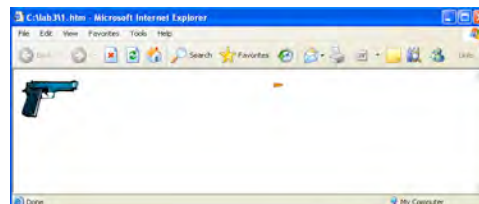
<body onKeyDown="shoot();b01.style.display='inline';">


</span>

</body>
</html>

```

The output looks:



Sometimes, the **while** statement and **if..then** statement produce the same results. You can write the Cat Running game using the while statement. For example,

```
<html>
<body onLoad=cc()>

<script>
var i=0;
function cc() {
m1.style.left=i;
i+=5;

while (i>=document.body.clientWidth-20) {i=0; }
setTimeout("cc()",70);
}
</script>


</body>
</html>
```

You can always use a *while* loop to run the same block of code while a specified condition is true.

The *do..while* loop

Many programming language also provides a do-while statement, which can be expressed as follows:

```
do {
    statement(s)
} while (expression);
```

The **do..while** statement is very similar to the while statement. For example,

```
function cancel_to_finish ( )
{
    var confirm_result;

    do
    {
        confirm_result = confirm ( "Press Cancel to finish!" );
    } while ( confirm_result != 0 );

}
```

The difference between do-while and while is that do-while evaluates its expression at the bottom of the loop instead of the top. Therefore, the statements within the do block are always executed at least once,

Use a do...while loop to run the same block of code while a specified condition is true. This loop will always be executed at least once, even if the condition is false, because the statements are executed before the condition is tested.

Review Questions

1. Which is a valid expression?  
A. `i = Math.floor(Math.random()*10);`  
B. `i = i + 2;`  
C. `i++;`  
D. `for (i=0; i<=10; i++)`

2. Given the following code block, which statement is correct?

```
if (i>=document.body.clientWidth-20) { i=i+1; }
```

- A. If this test evaluates to false, control jumps to { i=i+1; }
- B. It tests to determine if the value of variable i is larger or equal to player's web browser width.
- C. document.body.clientWidth represents the server's resolution values
- D. All of the above

3. Which is an example of "simple decisive logic"?

- A. if (i>= 3) { i++; } else { i--;}
- B. if (i>= 3) { i++; } else if { i--;} else {i=0;}
- C. if (i>= 3) { i++; }
- D. None of the above

4. Given the following code block, which statement is correct?

```
if (eval(m1.style.width.replace('px','')) < 400) {  
    setTimeout("cc()", 50);  
}
```

- A. The condition states that the value of width must be less than 400px.
- B. If m1's width value is >=400, call the cc() function.
- C. The condition states that it only allows 50 times of executions.
- D. If m1's width equals to 400 multiplied by 50, then stops.

5. Given the following, which is a keyword that must always be present?

```
m1.style.width
```

- A. m1
- B. style
- C. width
- D. all of the above

6. Given the following code segment, what does the replace() method do?

```
eval(m1.style.width.replace('px',''));
```

- A. It removes the 'px' substring.
- B. It replaces the 'px' substring with the user's entry.
- C. It replaces the 'px' substring with the new width value.
- D. It replaces the " substring with the 'px'.

7. Given the following code segment, which is the output if the value of i is 0?

```
swith(i)  
case 1: p1.innerHTML = "Apple"; break;  
case 2: p1.innerHTML = "Orange"; break;  
case 3: p1.innerHTML = "Banana"; break;  
default: p1.innerHTML = "Grape"; break;
```

- A. Apple
- B. Orange
- C. Banana
- D. Grape

8. Given the following code segment, which is not a possible output?

```
var i = Math.floor(Math.random()*3);
```

- A. 3
- B. 2
- C. 1
- D. 0

9. Given the following code segment, what will be inserted to p1?

```
for (i=0; i<=4; i++) {  
  p1.innerHTML += i;  
}
```

- A. 0123
- B. 01234
- C. 1234
- D. 123

10. Given the following code segment, Which statement is correct?

```
while (i >= 400) {  
  b01.style.display='none';  
  i=95;  
  b01.style.pixelLeft = i;  
}
```

- A. while the y-coordinate is greater than or equal to 400, set the y-coordinate value to 95
- B. while the y-coordinate is greater than or equal to 95, set the y-coordinate value to 400
- C. while the x-coordinate is greater than or equal to 400, set the x-coordinate value to 95
- D. while the x-coordinate is greater than or equal to 95, set the x-coordinate value to 400

## Game Programming

Lab #3

Control of flows

### Preparation #1:

1. Create a new directory named **C:\games**.
2. Use Internet Explorer to go to **<http://business.cypresscollege.edu/~pwu/cis261/download.htm>** to download lab3.zip (a zipped) file. Extract the files to C:\games directory.

### Learning Activity #1: Shooting gun

1. Change to the C:\games directory.
2. Use Notepad to create a new file named **C:\games\lab3\_1.htm** with the following contents:

```
<html>

<script>
function shoot() {

var i = b01.style.pixelLeft;

if (i <= document.body.clientWidth - b01.style.pixelWidth - 10) {
    b01.style.left=i + 10;
    setTimeout("shoot()",20);
}
else {b01.style.display='none';
    b01.style.left=95;
};
}
</script>

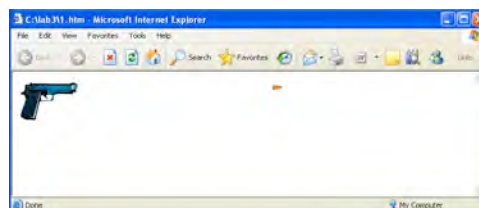
<body onKeyDown="shoot();b01.style.display='inline';">


</span>

</body>
</html>
```

Note: Compare this version with the one in lecture note. This version uses **if..then** statement; the other use **while loop**.

3. Test the program. Press any key to shoot. Watch how the bullet moves. A sample output looks:



### Learning Activity #2: UFO

1. Change to the C:\games directory.
2. Use Notepad to create a new file named **C:\games\lab3\_2.htm** with the following contents:

```
<html>
<style>
.dots {position:absolute; color:white}
```

```

</style>
<script>
code1 = "";
function draw_star() {
  for (i=1; i<=600; i++) {
    x = Math.round(Math.random()*screen.width);
    y = Math.round(Math.random()*screen.height);
    code1 += "<span class=dots id=dot"+i;
    code1 += " style='left:" + x + "; top:" + y + "'>.</span>";
  }
  bar.innerHTML = code1;
}

function ufo_fly() {
  ufo.style.left = Math.round(Math.random()*screen.width);
  ufo.style.top = Math.round(Math.random()*screen.height);
  setTimeout("ufo_fly()", 500);
}

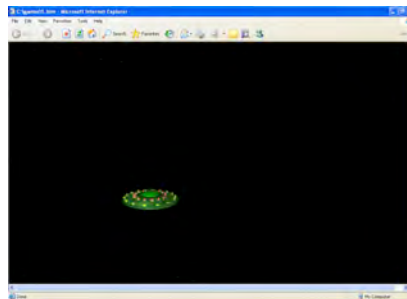
</script>

<body bgcolor=black onLoad="draw_star();ufo_fly()">
<div id="bar" style="z-index:0"></div>


</body>
</html>

```

3. Test the program. A UFO will appear and move around the sky that has 300 stars. A sample output looks:



### Learning Activity #3: Shuffle-Organize playing card

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab3\_3.htm with the following contents:

```

<html>

<script>
var code="";

function lineup(){
areal.innerText = "";

  for (i=1; i<=52; i++) {
    if (i%13 == 0) {
      code += "<img src=c" + i + ".gif id=c" + i + "><br>";
    }
    else {
      code += "<img src=c" + i + ".gif id=c" + i + ">";
    }
  }
}

```

```

areal.innerHTML = code;
code = "";
}

function scramble() {
areal.innerText = "";

    for (i=1; i<=52; i++) {

        x = Math.round(Math.random()*200);
        y = Math.round(Math.random()*200);
        code += "<img src=c" + i + ".gif id=c";
        code += i + " style='position:absolute; ";
        code += "left:" + x + "; top:" + y + "'>";
    }

areal.innerHTML = code;
code = "";
}

</script>

<body onLoad=scramble() onClick=scramble() onKeyDown=lineup()>

<span id="areal"></span>
</body>

</html>

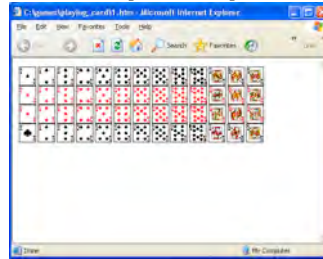
```

3. Test the program. Press any key to organize, click the mouse to shuffle. A sample output looks:

Before organizing



After organizing



#### Learning Activity #4: Moving the ball

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab3\_4.htm with the following contents:

```

<html>
<script>
function move() {
    var i = event.keyCode;
    switch(i) {
        case 37:

            if ( ball.style.pixelLeft <= 10) { ball.style.pixelLeft=10; }
            else { ball.style.pixelLeft -= 1; }
            break;

        case 39:
            if ((ball.style.pixelLeft + ball.style.pixelWidth) >= areal.style.pixelWidth-10)
            { ball.style.pixelLeft = (areal.style.pixelWidth - ball.style.pixelWidth -10); }
            else { ball.style.pixelLeft += 1; }
    }
}

```

```

        break;

    case 38:
        if ( ball.style.pixelTop <= 10) { ball.style.pixelTop=10; }
        else { ball.style.pixelTop -= 1;}
        break;





    case 40:
        if ((ball.style.pixelTop + ball.style.pixelHeight) >= areal.style.pixelHeight-
10)
        { ball.style.pixelTop = (areal.style.pixelHeight - ball.style.pixelHeight -10);}
        else { ball.style.pixelTop += 1; }
        break;
    }
}
</script>

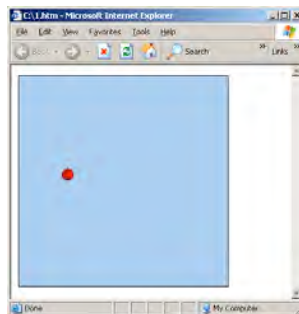
<body onkeydown=move()>
    <div id=areal style="border:solid 1 black;
        background-color:#abcdef;
        width:300px; height:300px; top:10;left:10;
        position:absolute"> </div>

</body>
</html>

```

3. Test the program. Use the , , ,  arrow keys to move the ball. A sample output looks:



### Learning Activity #5: Squeeze the clown

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab3\_5.htm with the following contents:

```

<html>

<script>
var k=0;

function goUp() {
    bar.style.display='inline';
    bar.style.pixelTop=bar.style.pixelTop - 5;

    if (clown.style.pixelTop <= 20) {
        clown.style.pixelHeight = bar.style.pixelTop - 20;
        clown.style.pixelWidth = clown.style.pixelWidth+1;
        clown.style.pixelTop = 20;
    }
}

```

```

else {
    bar.style.pixelHeight=k;
    k+=5;
    clown.style.pixelTop=bar.style.pixelTop-120;
}

}

</script>

<body onKeyDown=goUp() >

<hr style="position:absolute; left:10; top:0;
background-color:red; width:100;z-index:2" size="20px" align="left">



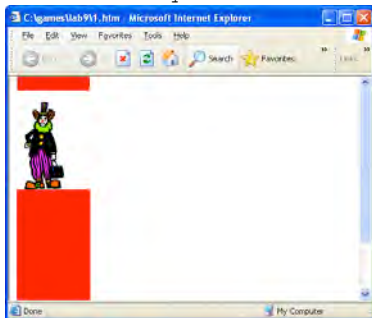
<span id="bar" style="position:absolute; left:10; top:400;
background-color:red; width:100; display:none; z-index:1"></span>

</body>
</html>

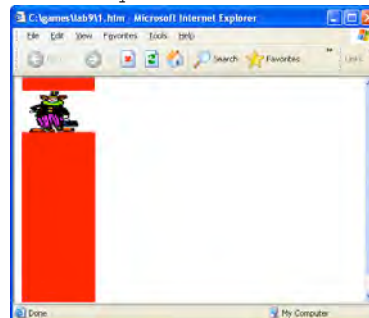
```

3. Test the program. Press the [Enter] key to pump up the clown. If the clown hit the ceiling, the clown gets squeezed. Two sample output are:

Clown not squeezed



Clown squeezed



### Submittal

Upon completing all the learning activities,

1. Upload all files you created in this lab to your remote web server.
2. Log in to Blackboard, launch Assignment 03, and then scroll down to question 11.
3. Copy and paste the URLs to the textbox. For example,
  - [http://www.geocities.com/cis261/lab3\\_1.htm](http://www.geocities.com/cis261/lab3_1.htm)
  - [http://www.geocities.com/cis261/lab3\\_2.htm](http://www.geocities.com/cis261/lab3_2.htm)
  - [http://www.geocities.com/cis261/lab3\\_3.htm](http://www.geocities.com/cis261/lab3_3.htm)
  - [http://www.geocities.com/cis261/lab3\\_4.htm](http://www.geocities.com/cis261/lab3_4.htm)
  - [http://www.geocities.com/cis261/lab3\\_5.htm](http://www.geocities.com/cis261/lab3_5.htm)

No credit is given to broken link(s).

## Game Programming

### Lecture #4 Event and Event Handler

**Introduction** The most dramatic switch for most people new to game programming is the fact that Windows-based game programs are event-driven, which means such game programs can respond to their operating system environment, as opposed to the environment taking cues from the program. In this lecture, you will learn to use event handlers to detect and respond to user's activity.

**What is an event?** When a user plays a game, any activity the user has is an **event**. Objects on the game can contain the so-called **event handler** that can trigger your functions to respond to such events. For example, you add an **onClick** event handler to trigger a function when the user clicks that button (an object).

```
<button onClick="myFunction()">Click Me</button>
```

In this case, the button is an object, and it has an event handler **onClick** that triggers a user-defined function named **myFunction()**.

During normal program execution, a large number of events occur, so skilled use of the event-oriented programming is very rewarding. It opens up possibilities for creating very intricate and complex game programs.

The DOM (Document Object Model) contains many objects that store a variety of information about the browser, the computer running the browser, visited URLs, and so on. You can use them to make your game program more joyful.

**The event object** According to DHTML, the **event** object represents the state of an event (meaning whenever the event occurs), the state of the keyboard keys, the location of the mouse, and the state of the mouse buttons. In other words, the **event** object stores data about events.

Whenever an event fires, the computer places appropriate data about the event into the event object - for example, where the mouse pointer was on the screen at the time of the event, which mouse buttons were being pressed at the time of the event, and other useful information.

Since the event object is a fairly active object, and it constantly changing its properties, you need to have a good handle of the even object. There are some general-purpose properties of the event object covered in this article are listed in the following table.

Event object's property	Description
srcElement	The element that fired the event
type	Type of event
returnValue	Determines whether the event is cancelled
cancelBubble	Can cancel an event bubble

#### The event.srcElement property

The srcElement and type properties contain the data that effectively encapsulates our object/event pair.

The srcElement property returns the element that fired the event. This is an object, and has the same properties as the element. For example, given an image object

```
<html><body>  
<img id='Image1' src='picture1.jpg'
```

```

onClick="p1.innerText=event.srcElement.id"
onDbClick="p1.innerText=event.srcElement.src"
onMouseOver="p1.innerText=event.srcElement.tagName">
<p id=p1></p>
</body></html>

```

When clicking on this image with an **id** attribute of 'Image1', and a **src** attribute of 'picture1.jpg', then **event.srcElement.id** will return 'Image1', and when double clicking on it the **event.srcElement.src** will return 'xxxxx/picture1.jpg'. Notice that the **xxxxx/** part will be extended because the computer internally converts relative URLs into absolute URLs.

Similarly, the **srcElement** has a **tagName** property:

```
event.srcElement.tagName
```

will return 'IMG'. And we can also read styles, so if the image has a style height of 100px, then **event.srcElement.style.height** will return '100px'.

### The event.type property

The **type** property displays the event name. If the event handler is the **onClick**, the type will be 'Click', and if the event is **onKeyPress**, then type will be 'KeyPress'.

```

<body
  onClick="status='You used the '+event.type+'
    event handler to click on me!'"
  onKeyPress="status='You used the '+event.type+'
    event handler to click on me!'">
</body>

```

There are some event properties designed for the mouse buttons. A detailed discussed is available in a later lecture.

clientX	Mouse pointer X coordinate relative to window
clientY	Mouse pointer Y coordinate relative to window
offsetX	Mouse pointer X coordinate relative to element that fired the event
offsetY	Mouse pointer Y coordinate relative to element that fired the event
button	Any mouse buttons that are pressed

For example, the following displays the current mouse position in the browser's status window.

```

<html><body onmousemove="status = '(' + event.x + ', ' +
  event.y + ')'">
</body></html>

```

There are also some event properties designed for the computer keyboard. A detailed discussed is available in a later lecture.

altKey	True if the alt key was also pressed
ctrlKey	True if the ctrl key was also pressed
shiftKey	True if the shift key was also pressed
keyCode	Returns UniCode value of key pressed

Consider the following. It displays a message only when the user press the Shift key.

```

<html>
<script>
function cc() {
  if (window.event.shiftKey)

```

```

        p1.innerText="Why did you press the Shift key?"
    }
</script>
<body onKeyDown="cc()">
<p id=p1></p>
</body></html>

```

As we proceed through this class, you will see how you can successfully employ the event objects to enhance your game programming.

Events and  
event handlers

**Events** are the user's activities, such as clicking a button, pressing a key, moving the mouse cursor around an object, and so on.

You can use the event handlers to detect a particular user activity, for example, clicking a mouse button, and call a function to render some results (such as an alert message) as response to the user's activity.

To allow you to run your bits of code when these events occur, JavaScript provides us with event handlers. All the event handlers in JavaScript start with the word on, and each event handler deals with a certain type of event. Here's a list of all the event handlers in JavaScript, along with the objects they apply to and the events that trigger them:

Table: Common Event Handlers

Event handler	Applies to:	Triggered when:
<b>onAbort</b>	Image	The loading of the image is cancelled.
<b>onBlur</b>	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea, Window	The object in question loses focus (e.g. by clicking outside it or pressing the TAB key).
<b>onCut</b>	Document	Fires on the source element when the object or selection is removed from the document and added to the system clipboard.
<b>onClick</b>	Button, Document, Checkbox, Link, Radio, Reset, Submit	The object is clicked on.
<b>onDblClick</b>	Document, Link	The object is double-clicked on.
<b>onDragDrop</b>	Window	An icon is dragged and dropped into the browser.
<b>onError</b>	Image, Window	A JavaScript error occurs.
<b>onPaste</b>	Document	Fires on the target object when the user pastes data, transferring the data from the system clipboard to the document.
<b>onKeyDown</b>	Document, Image, Link, TextArea	The user presses a key.
<b>onKeyPress</b>	Document, Image, Link, TextArea	The user presses or holds down a key.
<b>onKeyUp</b>	Document, Image, Link, TextArea	The user releases a key.
<b>onLoad</b>	Image, Window	The whole page has finished loading.
<b>onMouseDown</b>	Button, Document, Link	The user presses a mouse button.
<b>onMouseMove</b>	None	The user moves the mouse.

<b>onMouseOut</b>	Image, Link	The user moves the mouse away from the object.
<b>onMouseOver</b>	Image, Link	The user moves the mouse over the object.
<b>onMouseUp</b>	Button, Document, Link	The user releases a mouse button.
<b>onMove</b>	Window	The user moves the browser window or frame.
<b>onResize</b>	Window	The user resizes the browser window or frame.
<b>onUnload</b>	Window	The user leaves the page.

A complete list of event handler is available at <http://msdn.microsoft.com/workshop/author/dhtml/reference/events.asp?frame=true>.

To use an event handler, you usually place the event handler name within the HTML tag of the object you want to work with, followed by an assignment sign "=" and the associated JavaScript codes. For example:

```
<button onClick="alert('Thank You!')">Click Me</button>
```

Using events and event handlers in game programming

In a previous lecture, you tried the following code, which uses the event handler. You can now replace the event handler with anyone in the above table to test how they apply to your game scenario.

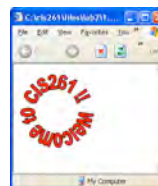
```
<html>
<script>
var i=2;

function cc() {
if (i>3) {i=1;}
m1.src="logo"+i+".gif";
i++;
}
</script>

<body onKeyDown="cc()">

</body>
</html>
```

An ideal sample output should look:



The truth is, as you probably already found out, not every event handler can fit in this game. You must choose the event handler wisely.

Consider the following. It uses the **onMouseDown** event handler to trigger the cc() function. Inside the cc() function, the button property of the event object (as in **event.button**) passes the mouse button value to the variable *i*. This value is then used to decide which .gif file to be displayed: 1 for car\_left.gif, and 2 for car\_right.gif.

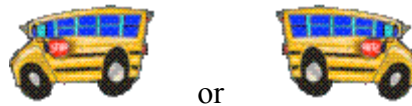
```

<html>
<script>
function cc() {
var i = event.button;
switch (i) {
case 1:
p1.innerHTML = "<img src='car_left.gif'>"; break;
case 2:
p1.innerHTML = "<img src='car_right.gif'>"; break;
}
}
</script>
<body onMouseDown=cc ()>

<p id=p1></p>
</body></html>

```

When clicking either the left or right mouse button, one of the .gif file appears:



The logic behind the scene is that the button property returns:

event.button value	Description
1	Left Mouse Button
2	Right Mouse Button
4	Middle Mouse Button

A detailed discussion about using mice as user input device is available at later lecture.

Where can you  
use event  
handlers?

Many games require the computer to trigger user functions. The question is when and where can you use the event handlers? Basically, there are two ways:

- **Automatic:** When the page (document) is opened, or a panel (an area) is opened, let the newly opened page/panel automatically triggers functions.
- **On-demand:** Create an object inside the page or panel, and embed the demanded event handler onto that object.

In the following code, there are two functions: **init()** and **spin()**. The statement,

```
<body onLoad=init()>
```

uses an **onLoad** event handler to trigger the **init()** function whenever the page is loaded or refreshed, because the <body>...</body> tags construct the contents of the body area of browser.

The statement,

```

<button OnMouseDown="spin()"
onMouseUp="clearTimeout (spinning)">Spin</button>

```

uses the **onMouseDown** event handler to trigger the spin() function ONLY when the player holds a mouse button. In other words, it is triggered on demand.

```

<html>
<script>
function init() {
card1.src=Math.floor(Math.random()*55)+".gif";

```

```

card2.src=Math.floor(Math.random()*55)+".gif";
}

function spin() {
card1.src=Math.floor(Math.random()*55)+".gif";
card2.src=Math.floor(Math.random()*55)+".gif";
spinning=setTimeout("spin()", 100);
}
</script>

<body onLoad=init()>

<table><tr>
  <td><img id=card1></td>
  <td><img id=card2></td>
</tr></table>

<button OnMouseDown="spin()"
  onMouseUp="clearTimeout(spinning)">Spin</button>

</body></html>

```

This code has access to 55 .gif files with file names of 0.gif, 1.gif, 2.gif, ..., and 54.gif. To randomly pick a .gif and assign it to **card1** (as in <img id=card1>, this code use:

```
card1.src=Math.floor(Math.random()*55)+".gif";
```

Each time when init() and spin() functions are trigger, a random number is generated and used to make up a file name for card1, such as *n*.gif, where *n* is the random number.

The last statement of the spin() function creates an object (named **spinning**), which is used to represent the setTimeout method. The spinning object triggers the spin() function every 100 milliseconds.

```
spinning=setTimeout("spin()", 100);
```

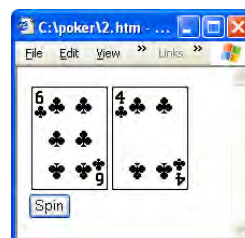
To end this loop, the following statement is used. The **onMouseUp** event handler triggers the **clearTimeout** method when the player release the mouse button.

```

<button OnMouseDown="spin()"
  onMouseUp="clearTimeout(spinning)">Spin</button>

```

When being executed, click the Spin button and hold the mouse, the cards spins till you release the mouse button.



Given the following HTML code which will load a GIF image file named **pencil.gif** with an ID **pen**.

```

<html>
<body>

```

```

    
  </body>
</html>

```

By adding the following CSS definition, this pencil.gif file will use absolute positioning system on the browser's body area, so it can be moved around with some JavaScript codes.

```

<style>
  img {position:absolute}
</style>

```

In order to move the pencil.gif file, we create a JavaScript function named **MovePenn()** containing two events: **event.clientX** and **event.clientY**.

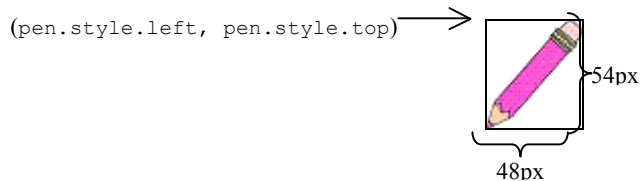
- The **clientX** property of event object sets or retrieves the x-coordinate of the mouse pointer's position relative to the client area of the window, excluding window decorations and scroll bars.
- The **clientY** property sets or retrieves the y-coordinate of the mouse pointer's position relative to the client area of the window, excluding window decorations and scroll bars.

```

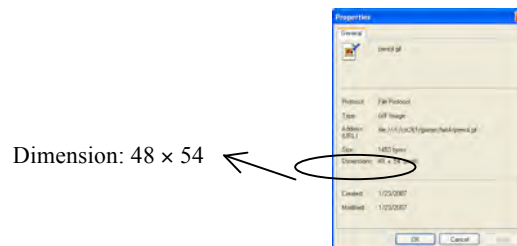
<script>
  function MovePen() {
    pen.style.left=event.clientX-30;
    pen.style.top=event.clientY-25;
  }
</script>

```

The absolute positioning system also uses the *x*- and *y*-coordinates to place the pencil.gif one the browser's body area. In the above code, we use **pen.style.left** to represent the *x*-coordinate, and **pen.style.top** to represent *y*-coordinate of the starting point of the pencil.gif file. The starting point of images is always defaulted to start with the upper-leftmost point.



The pencil.gif file has a dimension of 48px × 54px. You can detect the dimension using any image software (such as Paint) or even the Internet Explorer (by right clicking the image file and select Properties).



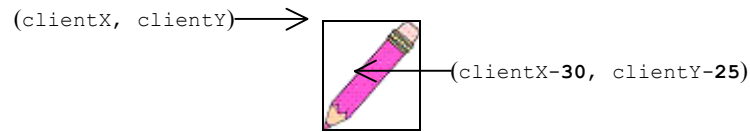
In order to move the pencil inside the pencil.gif file around the body area of browser with the mouse cursor, we need to continuously assign the current *x*- and *y*-coordinates of mouse cursor to **pen.style.left** and **pen.style.top**. The MovePen() function thus must contain the following code:

```

pen.style.left=event.clientX;
pen.style.top=event.clientY;

```

However, we want the mouse cursor to point at the pencil inside the pencil.gif file, it is necessary to adjust the values of event.clientX and event.clientY.



After a few trials, **(clientX-30, clientY-25)** seems to be a pair of ideal values (Note: You can use other pairs, too), so the code is modified to:

```
pen.style.left=event.clientX-30;
pen.style.top=event.clientY-25;
```

In JavaScript, as well as many other languages, a user-defined function must be called or it will not execute. You can use the **OnMouseMove** event handler to call the MovePen() function whenever the user attempts to move the pencil.

```

```

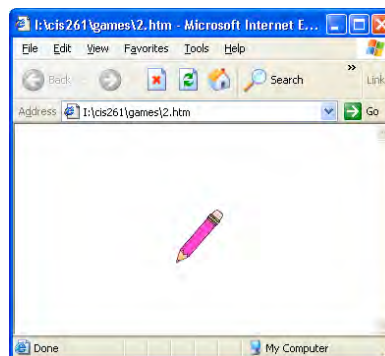
The complete code now looks:

```
<html>
<style>
  img {position:absolute}
</style>

<script>
function MovePen() {
pen.style.left=event.clientX-30;
pen.style.top=event.clientY-25;
}
</script>

<body>
  
</body>
</html>
```

When you test this code, you can move the pencil around the body area of the browser by first putting the mouse cursor on the pencil.gif file and then moving the cursor.



The event object and event handlers are very useful to the game programming, and technically can be applied to any aspects of game programming. For example, they can work with DHTML text appending methods, innerHTML, to draw scribble lines.

```

<html>

<style>
.dots {position:absolute;font-size:78}
</style>

<script>
code = "";
function cc() {
  x = event.clientX;
  y = event.clientY;

  code += "<span class=dots style='left: " + x + "; top: " + y
+ "'>.</span>";
  areal.innerText = "";
  areal.innerHTML = code;
}
</script>

<body onMouseMove=cc() style="cursor:arrow">
<div id=areal></div>
</body>
</html>

```

When being executed, you can move the mouse to draw free shape (scribble) lines.



**Drag and drop** Many games require the Drag-and-Drop function. There are several reasons you might want to incorporate this Drag-and-Drop ability into your games. One of the simplest reasons is to reorganize Data. As an example, you might want to have a queue of items that your users can reorganize. Instead of putting an input or select box next to each item to represent its order, you could make the entire group of items draggable. Or perhaps you want to have an object that can be moved around by your users.

"Moving object" is the topic of a later lecture. For now, you just have to know how the events and event handlers help to create the Drag-and-Drop ability; particularly how you can use event handlers to support this capacity so that the play can click on an item and drag it, and finally move the item.

To make your DHTML games support Drag-and-Drop, the following code must be used. It defines a "dragme" class.

```

var ie=document.all;
var nn6=document.getElementById&&!document.all;

var isdrag=false;
var x,y;
var dobj;

function movemouse(e)

```

```

{
  if (isdrag)
  {
    dobj.style.left = nn6 ? tx + e.clientX - x : tx +
event.clientX - x;
    dobj.style.top  = nn6 ? ty + e.clientY - y : ty +
event.clientY - y;
    return false;
  }
}

function selectmouse(e)
{
  var fobj      = nn6 ? e.target : event.srcElement;
  var topelement = nn6 ? "HTML" : "BODY";

  while (fobj.tagName != topelement && fobj.className !=
"dragme")
  {
    fobj = nn6 ? fobj.parentNode : fobj.parentElement;
  }

  if (fobj.className=="dragme")
  {
    isdrag = true;
    dobj = fobj;
    tx = parseInt(dobj.style.left+0);
    ty = parseInt(dobj.style.top+0);
    x = nn6 ? e.clientX : event.clientX;
    y = nn6 ? e.clientY : event.clientY;
    document.onmousemove=movemouse;
    return false;
  }
}

document.onmousedown=selectmouse;
document.onmouseup=new Function("isdrag=false");

```

To understand this code, you need to have a strong background in JavaScript and DHTML, but do not worry about it for now. It is displayed here to give you an ideal how the event and event handler model contribute to the game programming.

As to you, simply treat this code as a library code. You do so by saving it as an individual file with extension **.js**, such this file can be used as a library file. To call this library file from another DHTML file (e.g. lab4\_5.htm), simply add the following line between <head> and </head> of the lab4\_4.htm file.

```
<script src="dragndrop.js"></script>
```

Add the **class="dragme"** to whichever image you wish to make draggable. For example,

```

```

#### Review Questions

1. Given the following code, which is the name of the user-defined function?

```
<button id="click" onClick="Click()">Click Me</button>
```

- A. click
- B. onClick
- C. Click
- D. Click Me

2. Which is an example of the state of an event?

- A. the state of the keyboard keys
- B. the location of the mouse
- C. the state of the mouse buttons
- D. All of the above

3. Which event object's property determines whether the event is cancelled?

- A. SrcElement
- B. type
- C. returnValue
- D. cancelBubble

4. Given the following code block, what will be displayed on the status bar?

```
<img id='clown' src='clown.jpg'  
onClick="status=event.srcElement.id">
```

- A. clown
- B. clown.jpg
- C. IMG
- D. all of the above

5. Given the following code block, what will be displayed on the status bar?

```
<body onClick="status='You used the '+event.type+'  
event handler to click on me!'">
```

- A. Click
- B. onClick
- C. event.click
- D. event.onClick

6. Which is an example of event?

- A. clicking a button
- B. pressing a key
- C. moving the mouse cursor around an object
- D. All of the above

7. Which event handler is triggered when the object or selection is removed from the document and added to the system clipboard?

- A. onClick
- B. onCut
- C. onPaste
- D. onKeyPress

8. Which event handler is triggered when the user moves the browser window or frame?

- A. onWindowMove
- B. onMouseMove
- C. onDocumentMove
- D. onMove

9. Which triggers the init() function whenever the page is loaded or refreshed?

- A. <body onStart=init(>
- B. <body onLoad=init(>
- C. <body onInit=init(>
- D. <body onHold=init(>

10. How do you determinate the following code block?

```
objSpin=setTimeout("spin()", 100);
```

- A. clearTimeout(objSpin);
- B. clearTimeout("objSpin", 100);
- C. clear("objSpin", 100);
- D. clear(objSpin);

## Appendix A: More members exposed by the event object

MoreInfo	Retrieves the MoreInfo content of an entry banner in an ASX file through the event object.
nextPage	Retrieves the position of the next page within a print template.
propertyName	Sets or retrieves the name of the property that changes on the object.
qualifier	Sets or retrieves the name of the data member provided by a data source object.
reason	Sets or retrieves the result of the data transfer for a data source object.
recordset	Sets or retrieves from a data source object a reference to the default record set.
repeat	Retrieves whether the onkeydown event is being repeated.
returnValue	Sets or retrieves the return value from the event.
saveType	Retrieves the clipboard type when oncontentsave fires.
screenX	Retrieves the x-coordinate of the mouse pointer's position relative to the user's screen.
screenY	Sets or retrieves the y-coordinate of the mouse pointer's position relative to the user's screen.
srcFilter	Sets or retrieves the filter object that caused the onfilterchange event to fire.
srcUrn	Retrieves the Uniform Resource Name (URN) of the behavior that fired the event.
toElement	Sets or retrieves a reference to the object toward which the user is moving the mouse pointer.
type	Sets or retrieves the event name from the event object.
userName	Retrieves the sFriendlyName parameter that is passed to the useService method.
wheelDelta	Retrieves the distance and direction the wheel button has rolled.
x	Sets or retrieves the x-coordinate, in pixels, of the mouse pointer's position relative to a relatively positioned parent element.
y	Sets or retrieves the y-coordinate, in pixels, of the mouse pointer's position relative to a relatively positioned parent element.

## Game Programming

### Lab #4

### Event and Event Handler

#### Preparation #1:

1. Create a new directory named **C:\games**.
2. Use Internet Explorer to go to **<http://business.cypresscollege.edu/~pwu/cis261/download.htm>** to download lab4.zip (a zipped) file. Extract the files to C:\games directory.

#### Learning Activity #1: Drawing

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named **C:\games\lab4\_1.htm** with the following contents:

```
<html>
<style>
  img {position:absolute}
</style>

<script>
function MovePen() {
pen.style.left=event.clientX-30;
pen.style.top=event.clientY-25;
}
</script>

<body>
  
</body>
</html>
```

3. Test the program. A sample output looks:



#### Learning Activity #2: Grimace Geckos

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named **C:\games\lab4\_2.htm** with the following contents:

```
<html>

<style>
.aOut {position:absolute}
```

```

</style>

<script>

function popup() {

var k = Math.floor(Math.random()*4);
areal.innerText="";
for (i=1; i<=k; i++) {

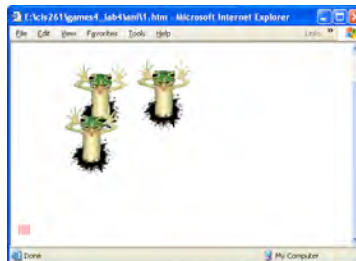
    var x = Math.floor(Math.random()*200);
    var y = Math.floor(Math.random()*200);
    codes = "<img class=aout id=geico" + i + " src='geico.gif'";
    codes += "style='left:" + x + "; top:" + y + "'"
    codes += " onClick=bar()>";
    areal.innerHTML += codes;
}
setTimeout("popup()", 1000)
}

function bar() {
area2.insertAdjacentHTML('BeforeBegin', '<b style=color:red>|</b>');
}

</script>
<body onLoad=popup()>
<div id=areal style="width:300;height:300"></div>
<div id=area2></div>
</body>
</html>

```

3. Test the program. When some geckos randomly appear, use the mouse click to hit them. A red bar grows each time when you hit a geico. A sample output looks:



### Learning Activity #3: An incomplete slot machine

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named **C:\games\lab4\_3.htm** with the following contents:

```

<html>

<script>

function init() {
for (i=1; i<=4; i++) {
    var k = Math.floor(Math.random()*55);
    areal.innerHTML += ""

```

```

    }
}

function spin() {
    areal.innerText="";
    for (i=1; i<=4; i++) {
        var k = Math.floor(Math.random()*55);
        areal.innerHTML += ""
    }
    spinning=setTimeout("spin()",100);
}

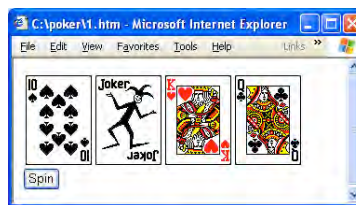
</script>
<body onLoad=init()>

<div id=areal></div>

<button onMouseDown=spin()
    onMouseUp=clearTimeout(spinning)>Spin</button>
</body>
</html>

```

3. Test the program. Click the Spin button and hold the mouse, the cards spins till you release the mouse button. A sample output looks:



#### Learning Activity #4: A simple Black Jack game

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named **C:\games\lab4\_4.htm** with the following contents:

```

<html>
<script>

function serve() {
    dealer1.src=Math.floor(Math.random()*52)+1+".gif";
    dealer2.src=Math.floor(Math.random()*52)+1+".gif";
    player2.src=Math.floor(Math.random()*52)+1+".gif";
}

function add_card() {
    p1.innerHTML+="*52)+1)+ )

```

```

<body onLoad=serve()>

<center><table width="300px">

<tr height="80px">
<tb>
<img id=dealer1>
<img id=dealer2>
</tb>

<tr height="40px"><td></td></tr>
</tr>
<tr height="80px">
<td>

<img id=player2>
<span id=p1></span>
</td>
</tr>

<tr height="40px">
<td>
<button onClick=add_card()>Add Card</button>
<button onClick="player1.src=Math.floor(Math.random()*52)+1+'.gif'">Reveal</button>
<button onClick=again()>Play Again</button>
</td>
</tr>

</table></center>

</body>
</html>

```

3. Test the program. When starting the game, you are given two cards. One card's value is not disclosed by default. Please Add Card if you want to add more card(s). Please Reveal to disclose the last card. A sample output looks:



### Learning Activity #5: Puzzle

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab4\_5.htm with the following contents:

```

<html>

<head>

<style>
.dragme{position:relative;}

```

```

td {border:solid 1 red; text-align:center; font-size:12px}
</style>

<script src="dragndrop.js"></script>

<script>

function cc() {

for (i=1;i<=12; i++) {

var x = Math.floor(Math.random()*50);
var y = Math.floor(Math.random()*100);

pic.innerHTML += "<img class='dragme' src=monalisa"+ i + ".gif style='left:" + x +
"; top:" + y + "'
ondragstart='this.style.zIndex=3;this.style.left=event.clientX;this.style.top=event
.clientY'>";

}

}

function dd() {
tb.style.display='none';
pic.innerText='';
for (k=1;k<=12; k++) {

if (k%3 == 0) {
pic.innerHTML+="<img src=monalisa" + k + ".gif style='border: solid 1 white'><br>";
}
else {
pic.innerHTML+="<img src=monalisa" + k + ".gif style='border: solid 1 white'>";
}
}
}
</script>
</head>

<body onLoad=cc()>
<button onClick=dd()>Solve</button>

<table border=0 cellpadding=0 id=tb style="position:relative;">
<tr height="52px">
<td width="60px">1</td>
<td width="64px">2</td>
<td width="60px">3</td>
</tr>

<tr height="94px">
<td>4</td>
<td>5</td>
<td>6</td>
</tr>

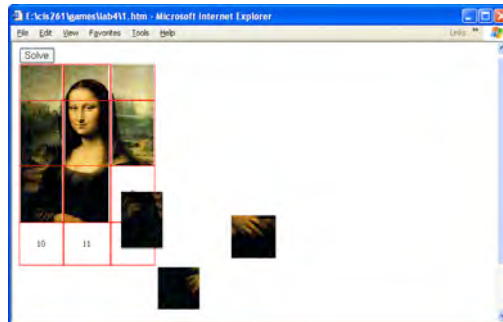
<tr height="82px">
<td>7</td>
<td>8</td>
<td>9</td>
</tr>

```

```
|  |  |  |
| --- | --- | --- |
| 10 | 11 | 12 |

```

3. Test the program. Move each piece to its expected cell. A sample output looks:



### Submittal

Upon completing all the learning activities,

1. Upload all files you created in this lab to your remote web server.
2. Log in to Blackboard, launch Assignment 04, and then scroll down to question 11.
3. Copy and paste the URLs to the textbox. For example,
  - [http://www.geocities.com/cis261/lab4\\_1.htm](http://www.geocities.com/cis261/lab4_1.htm)
  - [http://www.geocities.com/cis261/lab4\\_2.htm](http://www.geocities.com/cis261/lab4_2.htm)
  - [http://www.geocities.com/cis261/lab4\\_3.htm](http://www.geocities.com/cis261/lab4_3.htm)
  - [http://www.geocities.com/cis261/lab4\\_4.htm](http://www.geocities.com/cis261/lab4_4.htm)
  - [http://www.geocities.com/cis261/lab4\\_5.htm](http://www.geocities.com/cis261/lab4_5.htm)

No credit is given to broken link(s).

## Game Programming

### Lecture #5 Using mouse buttons for input control

**Introduction** The players interact with a game by giving user inputs. To most games, user inputs encompass the entire communications between a player and a game. There are many user input devices that are used in game programming, such as keyboard, mice, joysticks, flight sticks, touchpad, pointing devices, and many other user input devices have brought extended input capabilities to the game player; however, none is as popular as the mouse.

This lecture discusses the basic concepts of user input handling, but it will focus on the mouse. A later lecture will discuss the keyboard in details.

**Common User Input Devices** Input devices are the physical hardware that allows a user to interact with a game. Input devices all perform the same function: converting information provided by the user into a form understandable by the computer. Input devices form the link between the user and your game. Even though you can't directly control the input device hardware, you can certainly control how it is interpreted in your game. Currently, there are three primary types of user input devices:

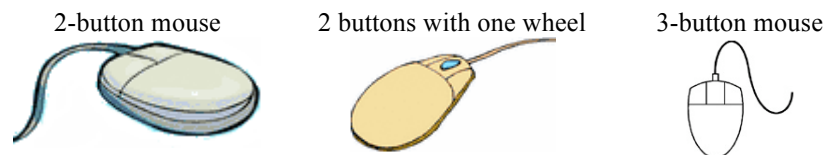
- The keyboard.
- The mouse.
- Joysticks.

**Why the mouse?** Operating system such as Windows, Linux, and Macintosh have all adopt the mouse as a standard input device, so it is a nature for the mouse to be the most popular user input device for game programming.

The mouse, however, does not share the wide range of input applications to games that the keyboard has. A mouse usually has 2 or 3 buttons, or 2 buttons with one wheel, while a computer keyboard has at least 101 keys. Additionally, the mouse was primarily designed as a point-and-click device. The truth is many games do not follow the point-and-click paradigm.

Surprisingly the mouse has the mobility to quickly move around a given area, so it does have a recognizable usefulness, which is dependent totally on the type of game and the type of user interaction dictated by the game. It is very important to learn how to handle user inputs with the mouse.

**Mouse Anatomy** When you move the mouse, a series of events is set off, and most computer languages provide a series of mouse messages that are used to convey mouse events to recognizable actions. The previous lecture discussed how the events and event handlers work closely together to detect and respond to user's activities. You should now try to understand how a mouse works so you can have a better manipulation of them in your games.



A mouse is usually equipped with 1-3 buttons. Each button is an individual interface that sends signals to the computer. A mouse is commonly used to make selections and position the cursor, but, in a graphical user interfaces environment, the mouse is also used to trace and record x- and y-coordinates of any given objects.

**Mouse Event** JavaScript includes a number of event handlers for detecting mouse actions. Your game script can

## Handlers

use them detect the movement of the mouse pointer and when a button is clicked, released, or both. They are:

- **onMouseOver**: The mouse is moved over an element.
- **onMouseOut**: The mouse is moved off an element.
- **onMouseMove**: The mouse is moved.
- **onMouseDown**: A mouse button is pressed.
- **onMouseUp**: A mouse button is released.
- **onClick**: When a mouse button is clicked.
- **onDbClick**: When a mouse button is double clicked.

### The onMouseOver event handler

The **onMouseOver** handler is called when the mouse pointer moves over a link or other object. In the following example, the **onMouseOver** handler tells the computer to change the image file's source from **gopher.gif** to **gopher2.gif** when the play mouse over the image file. The keyword "this" is used in the **this.src='gopher2.gif'** statement simply because the **onMouseOver** handler is used inline with the **<img>** tag.

```

```

The **onMouseOut** handler is the opposite—it is called when the mouse pointer moves out of the object's border. Unless something strange happens, this always happens some time after the **onMouseOver** event is called.

```

```

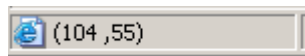
In the above example, the **onMouseOut** handler changes the file source from **gopher2.gif** back to **gopher.gif**. Guess what? This is a very commonly used trick used to create a visual effect of an object.

### TheonMouseMove event handler

The **onMouseMove** event occurs any time the mouse pointer moves. As you might imagine, this happens quite often—the event can trigger hundreds of times as the mouse pointer moves across a page. For example,

```
<body onMouseMove="status
=' ('+event.clientX+' , '+event.clientY+' ) ' "></body>
```

When moving the mouse cursor around, a pair of *x*- and *y*-coordinate appears on the status bar in the form of (*x*, *y*). Noticeably, the value of *x* and *y* will continuously change as response to the mouse's movement.



### The onMouseDown and onMouseUp event handler

To give you even more control of what happens when the mouse button is pressed, two more events are included:

- **onMouseDown** is used when the user presses the mouse button.
- **onMouseUp** is used when the user releases the mouse button.

Refer to the keyboard code of a previous lecture. You can apply **onMouseDown** and **onMouseUp** methods to it. For example,

```
<span class="wKey" id="midC" style="left:50" onMouseDown="CDown()"
onMouseUp="CUp()"></span>
<span class="wKey" id="midD" style="left:91" onMouseDown="DDown()"
```

```
onMouseUp="DUp () "></span>
<span class="wKey" id="midE" style="left:132"
onMouseDown="EDown () " onMouseUp="EUp () "></span>
```

### The **onClick** and **onDbClick** event handler

The **onMouseDown** and **onMouseUp** event handlers are the two halves of a mouse click. You can also use events to detect when the mouse button is clicked. The basic event handler for this is **onClick**. This event handler is called when the mouse button is clicked while positioned over the appropriate object.

The **onDbClick** event handler is similar to **onClick**, but is only used if the user double-clicks on an object. In the following example, **onClick** triggers the **cc()** function, while **onDbClick** triggers **dd()** function.

```
<button onClick="cc () " onDbClick="dd () ">Rotate</button>
```

If you want to detect an entire click, use **onClick**. Use **onMouseUp** and **onMouseDown** to detect just one or the other.

The event object's properties for mouse

Internet Explorer supports JavaScript's event object, and it provides the following common properties for record information of mouse movement. The data they record can be retrieved as reference to your game if you know how to use them:

- **event.button**: The mouse button that was pressed. This value is 1 for the left button and usually 2 for the right button.
- **event.clientX**: The x-coordinate (column, in pixels) where the event occurred.
- **event.clientY**: The y-coordinate (row, in pixels) where the event occurred.

The **event.clientX** property returns the horizontal coordinate within the application's client area at which the event occurred (as opposed to the coordinates within the page). For example, clicking in the top-left corner of the client area will always result in a mouse event with a **clientX** value of 0, regardless of whether the page is scrolled horizontally.

Similarly, **event.clientY** returns the horizontal coordinate within the application's client area at which the event occurred (as opposed to the coordinates within the page). For example, clicking in the top-left corner of the client area will always result in a mouse event with a **clientX** value of 0, regardless of whether the page is scrolled horizontally.

Syntax is:

```
event.clientX;
```

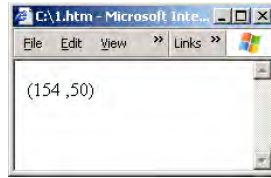
and

```
event.clientY
```

For example,

```
<html>
<body onMouseMove="p1.innerText
=' ('+event.clientX+' , '+event.clientY+') '";>
<p id=p1></p>
</body>
</html>
```

When executing the code, you will see a (x, y) coordinate set in the body area.



Consider the following code. It creates a red area starting at the point (100px, 100px) with a width of 100px and height of 50px.

```
<html>
<style>
.ar {position:absolute; left:100px; top:100px;
    width:100px; height:50px;
    background-color:red; border:solid 1 red}
</style>

<body onMouseMove=cc()>
<div class=ar id=areal></div>
</body>
</html>
```

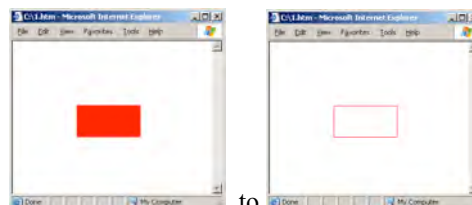
By adding the following bold lines, which means “when the mouse cursor’s y-coordinate between 100px and 150px change the background color to white; otherwise, set the background color to red”.

```
<html>
<style>
.ar {position:absolute; left:100px; top:100px;
    width:100px; height:50px;
    background-color:red; border:solid 1 red}
</style>

<script>
function cc() {
    if ((event.clientY >= 100) && (event.clientY <= 150)) {
        areal.style.backgroundColor="white";
    }
    else {
        areal.style.backgroundColor="red";
    }
}
</script>

<body onMouseMove=cc()>
<div class=ar id=areal></div>
</body>
</html>
```

Notice that the **onMouseMove** handler is placed inside the <body> tag, which means “when the mouse is moved around the body area”.



The **event.button** property is used to determine which mouse button has been clicked. To safely detect a mouse button you have to use the **onMouseDown** or **onMouseUp** events.

In the previous lecture, there was an explanation about how you could use the **onMouseDown** event handler to trigger the `cc()` function in the following example. This code is also a good example for the **event.button** property.

Again, the `event.button` property is designed to set or retrieve the mouse button pressed by the user. Possible values of the **event.button** property for Internet Explorer are:

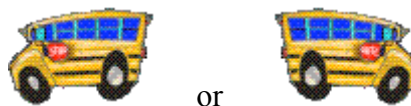
0	Default. No button is pressed.
1	Left button is pressed.
2	Right button is pressed.
3	Left and right buttons are both pressed.
4	Middle button is pressed.
5	Left and middle buttons both are pressed.
6	Right and middle buttons are both pressed.
7	All three buttons are pressed.

Inside the `cc()` function, the `button` property of the event object (as in **event.button**) passes the mouse button value to the variable *i*. This value is then used to decide which .gif file to be displayed: 1 for `car_left.gif`, and 2 for `car_right.gif`.

```
<html>
<script>
function cc() {
var i = event.button;
switch (i) {
case 1:
    p1.innerHTML = "<img src='car_left.gif'>"; break;
case 2:
    p1.innerHTML = "<img src='car_right.gif'>"; break;
}
}
</script>
<body onMouseDown=cc() onContextMenu="return false">

<p id=p1></p>
</body></html>
```

When clicking either the left or right mouse button, one of the .gif file appears:



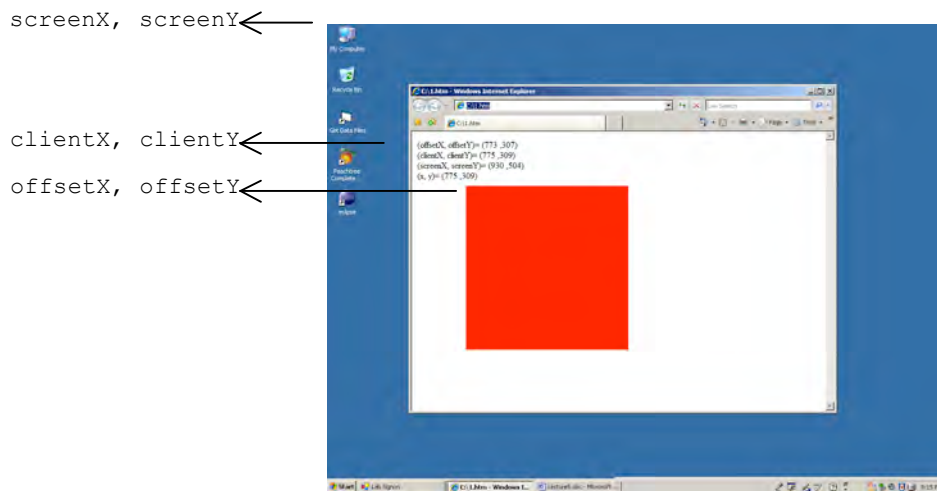
In this case, the `event.button` returns a numerical value to the computer. The computer in turn gives it to the variable *i* for evaluation. Notice that this property is read/write. The property has a default value of 0. This property is used with the **onmousedown**, **onmouseup**, and **onmousemove** events. For other events, it defaults to 0 regardless of the state of the mouse buttons.

In addition to **clientX**, **clientY**, and **button** properties, Internet Explorer also works with the following properties of the event object.

<code>event.offsetX</code>	sets or retrieves the x-coordinate of the mouse pointer's position relative to the object firing the event.
----------------------------	---

event.offsetY	Sets or retrieves the y-coordinate of the mouse pointer's position relative to the object firing the event.
event.screenX	sets and retrieves the x-coordinate of the mouse pointer's position relative to the user's screen.
event.screenY	sets or retrieves the y-coordinate of the mouse pointer's position relative to the user's screen.
event.x	sets or retrieves the x-coordinate, in pixels, of the mouse pointer's position relative to a relatively positioned parent element. Technically, this property is similar to clientX.
event.y	sets or retrieves the y-coordinate, in pixels, of the mouse pointer's position relative to a relatively positioned parent element. Technically, this property is similar to clientY.

The following figure illustrates the starting point of these properties.



The following example demonstrates how these properties are different from each other. Be sure to run this code to see for yourself.

```
<html>
<script>
function cc() {

p1.innerHTML = "(offsetX, offsetY)= (" + event.offsetX + " ," +
event.offsetY +")";
p2.innerHTML = "(clientX, clientY)= (" + event.clientX + " ," +
event.clientY +")";
p3.innerHTML = "(screenX, screenY)= (" + event.screenX + " ," +
event.screenY +")";
p4.innerHTML="(x, y)= (" + event.x + " ," + event.y +")";
}

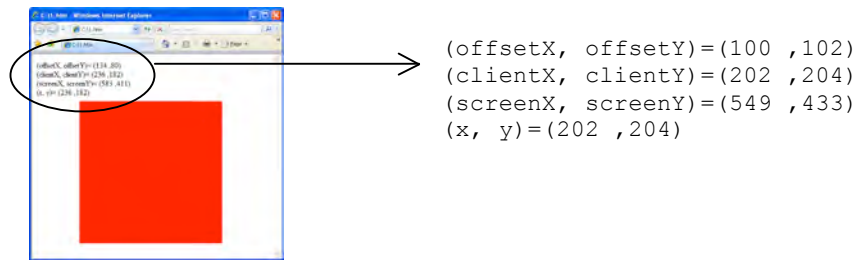
</script>
<span id=p1></span><br>
<span id=p2></span><br>
<span id=p3></span><br>
<span id=p4></span><br>
<body onMouseMove=cc()>

<div style="position:absolute; width:300;height:300;
left:100;top:100;background-color:red"></div>

</body>
```

```
</html>
```

When executing this code, move the cursor around the red area and compare the x- and y-coordinates of them.

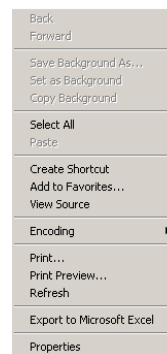


The difference between (offsetX, offsetY) and (clientX, clientY) was caused by the following setting of the above code, which sets the starting point of the red area as (100, 100), while the starting point of the browser's body area remains being (0, 0):

```
left:100;top:100;
```

Disable right click

One annoying feature of the GUI (graphical user interfaces) is its support of mouse right click, especially when the right click is used to trigger some menu. For example, when you right click on a blank space of the browser's body area, the following menu pops up.



This menu may disgrace your game programs, so many programmers choose to disable it. To do so, simply add the bold section to the <body> tag.

```
<body onContextMenu="return false">
```

Test the code in Internet Explorer 5.0 or later. When you attempt to right-click anywhere on the page, the shortcut menu is not displayed.

Another solution is to redefine what the right click should do. For example, in the following code, the right click is now used to trigger the **new\_right\_click()** function.

```
<html>
<script language="JavaScript">

function new_right_click() {
gl.style.display="inline";
}

function click(e) {
  if (document.all) {
    if (event.button == 2) {
```

```

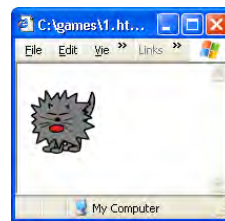
        new_right_click();
        return false;
    }
}
if (document.layers) {
    if (e.which == 3) {
        new_right_click();
        return false;
    }
}
}

if (document.layers) {
    document.captureEvents(Event.MOUSEDOWN);
}
document.onmousedown=click;
</script>

<body>
<img id=g1 src=cat.gif style="display:none">
</body></html>

```

Execute the code, and right click the mouse. The cat appears, but the above menu won't.



























The cursor shape

With CSS, you can change the value of the cursor declaration to use different cursor styles to enhance the appearance of your game. The syntax is:

```
element { cursor: Value }
```

where *element* is any HTML tag or user-declared style name, *cursor* is a keyword, and *value* is the anyone from the following list.

Look	Values	Example
	default	cursor:default
	crosshair	cursor:crosshair
	hand	cursor:hand
	pointer	cursor:pointer
	Cross browser	cursor:pointer;cursor:hand
	move	cursor:move
	text	cursor:text
	wait	cursor:wait
	help	cursor:help
	n-resize	cursor:n-resize

	ne-resize	cursor:ne-resize
	e-resize	cursor:e-resize
	se-resize	cursor:se-resize
	s-resize	cursor:s-resize
	sw-resize	cursor:sw-resize
	w-resize	cursor:w-resize
	nw-resize	cursor:nw-resize
	progress	cursor:progress
	not-allowed	cursor:not-allowed
	no-drop	cursor:no-drop
	vertical-text	cursor:vertical-text
	all-scroll	cursor:all-scroll
	col-resize	cursor:col-resize
	row-resize	cursor:row-resize

For example, when you mouse the cursor on top of the image file, the cursor turns into a cross.

```

```

You can even use your own custom images as cursors. The syntax is:

```
element { cursor : url("FileName"), value }
```

Notice that the **.cur** extension refers to the file extension for the Windows Cursor file. This is one format that cursors (the mouse pointer design) can be stored in, under Microsoft Windows 3.x and later. In Windows 95, NT and later, this format has been largely superseded by the ANI format which allows animated and color cursor designs.

The CUR format also allows transparency so that cursors do not have to appear rectangular when displayed on screen. For example,

```
body { cursor : url("custom.cur"), pointer }
```

Note: The custom images as cursors is only supported in Internet Explorer 6.0 or later, which is why we also included pointer, so if the browser dose not support custom cursors at least the default pointer will be displayed.

In the following game, the mouse cursor is changed to a graphic of pen, so it creates a visual effect of writing on the browser's body area using a pen.

```

<html>

<style>
.dots {position:absolute;font-size:30}
body {
  cursor : url("pencil.cur"), pointer
}

</style>

<script>

function cc() {
  x = event.clientX;
  y = event.clientY;

  codes = "<span class=dots style='left: " + x;
  codes += "; top: " + y +"'>.</span>";
  areal.innerHTML += codes;
}

</script>

<body onMouseMove=cc()>
<div id=areal></div>
</body>
</html>

```

A sample output looks:



#### Review Questions

- Which is the event handler that functions only when the mouse is moved off an element?
  - onMouseOver
  - onMouseOut
  - onMouseMove
  - onMouseDown
- The onMouseDown and onMouseUp together makes the function of \_\_\_.
  - onMouseClicked
  - onClick
  - onKeyClick
  - onButtonClick
- Given the following code block, which statement is correct?
 

```

```

  - The onMouseOver handler change the image file's source from gopher2.gif to gopher.gif.
  - The onMouseOver handler change the image file's source from gopher.gif to gopher2.gif.

- C. The word "this" in this.src='gopher2.gif' refers to the gopher2.gif image file  
 D. onMouseOver must be replaced with onMouseMove to make this code work.

4. Which code block is equivalent to the following one?

```
<body onMouseMove="status=
'+event.clientX+', '+event.clientY+'";></body>
```

- A. <body onMouseMove="status='('+event.X+', '+event.Y+')'";></body>  
 B. <body onMouseMove="status='('+client.X+', '+client.Y+')'";></body>  
 C. <body onMouseMove="status='('+X+', '+Y+')'";></body>  
 D. <body onMouseMove="status='('+clientX+', '+clientY+')'";></body>

5. Which statement about event.button is correct?

- A. This value is 0 for the left button and usually 1 for the right button.  
 B. This value is 1 for the left button and usually 2 for the right button.  
 C. This value is 0 for the left button and usually 2 for the right button.  
 D. This value is 1 for the left button and usually 4 for the right button.

6. Which sets or retrieves the y-coordinate of the mouse pointer's position relative to the object firing the event?

- A. event.offsetY  
 B. event.clientY  
 C. event.screenY  
 D. event.Y

7. Which can disable the right click function?

- A. <body onMenuClick="return false">  
 B. <body onRightClick="return false">  
 C. <body onContextMenu="return false">  
 D. <body onContextClick="return false">

8. Which is the correct way to change the mouse cursor to a question mark?

- A. cursor:wait  
 B. cursor:question  
 C. cursor:mark  
 D. cursor:help

9. The .cur file extension refers to \_\_\_\_.

- A. Windows Cursor file  
 B. Windows icon file  
 C. Windows graphic file  
 D. Windows screen saver file

10. In Internet Explorer, which value of event.button indicates all three buttons are pressed?

- A. 7  
 B. 6  
 C. 5  
 D. 3

## Game Programming

### Lab #5

Using mouse buttons for input control

#### Preparation #1:

1. Create a new directory named **C:\games**.
2. Use Internet Explorer to go to **<http://business.cypresscollege.edu/~pwu/cis261/download.htm>** to download lab5.zip (a zipped) file. Extract the files to C:\games directory.

#### Learning Activity #1: Using custom mouse cursor

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named **C:\games\lab5\_1.htm** with the following contents:

```
<html>

<style>
.dots {position:absolute;font-size:30}
body {
  cursor : url("pencil.cur"), pointer
}

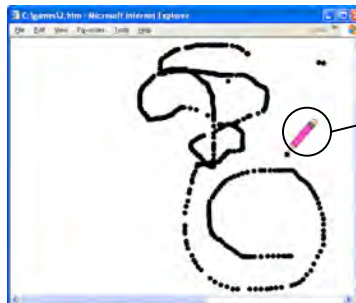
</style>

<script>
code = "";
function cc() {
  x = event.clientX;
  y = event.clientY;

  code += "<span class=dots style='left: " + x + "; top: " + y + "'>.</span>";
  areal.innerText = "";
  areal.innerHTML = code;
}
</script>

<body onMouseMove=cc()>
<div id=areal></div>
</body>
</html>
```

3. Test the program. A sample output looks:



The cursor changes to a pencil

#### Learning Activity #2: Angry cat

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

4. Change to the C:\games directory.
5. Use Notepad to create a new file named C:\games\lab5\_2.htm with the following contents:

```
<html>

<script>
function anger() {
cat.src='cat.gif'
setTimeout("cat.src='cat1.gif'", 1000);

cat.style.pixelLeft = Math.floor(Math.random() * (document.body.clientWidth -
cat.style.pixelWidth));

cat.style.pixelTop = Math.floor(Math.random() * (document.body.clientHeight -
cat.style.pixelHeight));
}
</script>

<div id=areal>
<img id="cat" src=cat1.gif onMouseOver="anger()" style="position:absolute">
</div>

</html>
```

6. Test the program. Move the cursor to approach the cat, the cat jumps and get angry. A sample output looks:

Original



Angry cat



### Learning Activity #3: A simple Tic-Tac-Toe

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab5\_3.htm with the following contents:

```
<html>
<head>
<style>
.dragme {position:relative;}
area2, area3 { position:absolute }
</style>

<script src="dragndrop.js"></script>
<script>
function cc() {
codes = "<table border=1 cellspacing=0>";
for (i=1; i<=3; i++) {
```

```

codes += "<tr height=54><td width=54>&nbsp;</td>";
codes += "<td width=54>&nbsp;</td>";
codes += "<td width=54>&nbsp;</td></tr>";
}
codes += "<table>";
area1.innerHTML= codes;
}

function insertx() {
    area2.innerHTML += "<img src=x.gif class='dragme'>";
}

function inserto() {
    area3.innerHTML += "<img src=o.gif class='dragme'>";
}

</script>

</head>

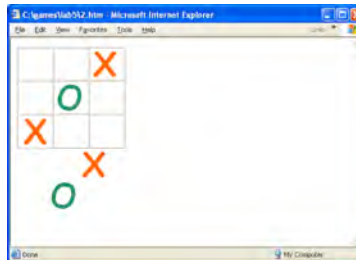
<body onLoad="cc()">

<span id=area1></span>

<div id=area2 onMouseUp=insertx()><img id=x0 src=x.gif class=dragme></div>
<div id=area3 onMouseUp=inserto()><img id=o0 src=o.gif class=dragme></div>
</span>
</body>
</html>

```

3. Test the program. A sample output looks:



#### Learning Activity #4: A simple Roulette wheel

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game. **A later lecture will explain how the ball can move in a circle.**

1. Change to the C:\games directory.
2. Use Notepad to create a new file named **C:\games\lab5\_4.htm** with the following contents:

```

<html>

<script>
var r=120; a=0;
function cc() {
    b1.style.left=197+r*Math.sin(a);
    b1.style.top=197+r*Math.cos(a);

    if (a >= Math.PI *2) { a=0; }
    else { a += Math.PI/180; }
}

```

```

rotating=setTimeout("cc()", 2);
}
</script>

<body>

<img src=wheel.gif style="position:absolute;
  z-index:0; left:0; top:0">

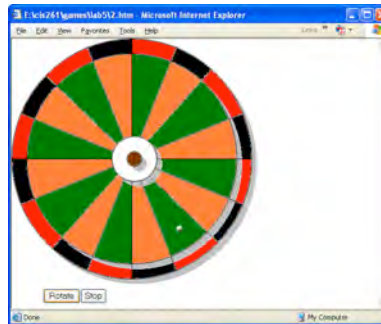
<div style="position:absolute; left:50; top:400;">
<button onClick="cc()">Rotate</button>
<button onClick=clearTimeout(rotating)>Stop</button>
</div>

</body>

<html>

```

3. Test the program. Click the Rotate button to start, and Stop to stop. A sample output looks:



### Learning Activity #5: Mini Monopoly Game

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Copy the lab5\_5.htm (the one you downloaded previously) to the C:\games directory.
3. Use Notepad to open the C:\games\lab5\_5.htm file, and add the following bold lines between <html> and <body> tags. (this part of code import the dragdrop.js library file and a function dice()):

```

<html>

<head>

<style>
.dragme{position:relative;}
td { font-size:11px;text-align:center;font-family:arial}
.cards {position:absolute; width:120; height:60;
border:solid 1 black; text-align:center; font-size:16px;
color:white; }
</style>

<script>

function init() {
for (i=1;i<=2;i++) {

```

```

var k=Math.floor(Math.random()*6)+1;
areal.innerHTML += "<img src=d"+k+".gif border=2> ";
}
}

function dice() {
areal.innerText="";
for (i=1;i<=2;i++) {
var k=Math.floor(Math.random()*6)+1;
areal.innerHTML += "<img src=d"+k+".gif border=2> ";
}
tossing=setTimeout("dice()", 50);
}

function chancewShow() {
cchance.style.backgroundColor="white";
cchance.style.fontSize="10px";
cchance.style.color="black";
var j = Math.floor(Math.random()*3);
switch (j) {
case 0: cchance.innerText="You won $50!"; break;
case 1: cchance.innerText="You lost $100!"; break;
case 2: cchance.innerText="Speeding fine, $20!"; break;
}
}

function chestShow() {
cchest.style.backgroundColor="white";
cchest.style.fontSize="10px";
cchest.style.color="black";
var j = Math.floor(Math.random()*3);
switch (j) {
case 0: cchest.innerText="Christmas donation, $50!"; break;
case 1: cchest.innerText="State Tax $100!"; break;
case 2: cchest.innerText="You won the lottery of $20!"; break;
}
}

function cover() {
cchance.style.backgroundColor="red";
cchance.style.fontSize="16px";
cchance.style.color="white";
cchance.innerText="Chance";

cchest.style.backgroundColor="green";
cchest.style.fontSize="16px";
cchest.style.color="white";
cchest.innerText="Community Chest";
}
</script>

<script src="dragndrop.js"></script>
</head>

```

```

</body>

```

4. Replace the <body> tag with the following line:

```

<body onLoad=init()>

```

5. At the end of the code, add the following bold lines between </table> and </body> tags:

```

</table>

```

```

</td><td>
<span id=areal></span>

<div class="cards" id="cchance" style="left:200; top:150;
  background-Color:red;" onClick="chancewShow()">Chance</div>

<div class="cards" id="cchest" style="left:310; top:270;
  background-Color:green;" onClick="chestShow()">Community Chest</div>

<button onMouseDown="dice();cover()" onMouseUp=clearTimeout(tossing)>Toss
Dice</button>
</td></tr></table>

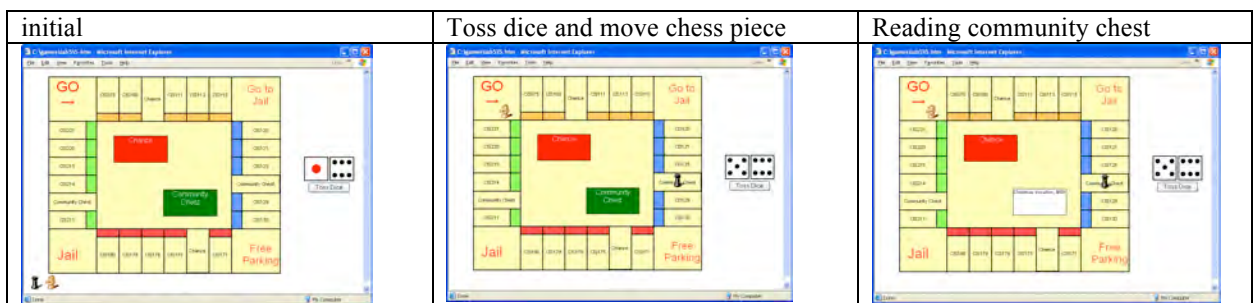



</body>

</html>

```

- Test the program. To play, first move the two chess pieces to the Go cell. Toss the dice, calculate the points, and move forwards. Click either Chance or Community block when you arrive at one of them. A sample output looks:



### Submittal

Upon completing all the learning activities,

- Upload all files you created in this lab to your remote web server.
- Log in to Blackboard, launch Assignment 05, and then scroll down to question 11.
- Copy and paste the URLs to the textbox. For example,
  - [http://www.geocities.com/cis261/lab5\\_1.htm](http://www.geocities.com/cis261/lab5_1.htm)
  - [http://www.geocities.com/cis261/lab5\\_2.htm](http://www.geocities.com/cis261/lab5_2.htm)
  - [http://www.geocities.com/cis261/lab5\\_3.htm](http://www.geocities.com/cis261/lab5_3.htm)
  - [http://www.geocities.com/cis261/lab5\\_4.htm](http://www.geocities.com/cis261/lab5_4.htm)
  - [http://www.geocities.com/cis261/lab5\\_5.htm](http://www.geocities.com/cis261/lab5_5.htm)

No credit is given to broken link(s).

## Game Programming

### Lecture #6 Using keyboard for input control

**Introduction** The keyboard has been the computer input device for the longest time, even since the old days when there was no such thing called mouse and graphical user interface. As a matter of fact, even today, the keyboard is still a useful input device for a wide range of games. For example, any game involving the player moving an object around will benefit from using the arrow keys. In this lecture, you will learn many games that uses the arrow keys to move objects.

**Basic keyboard-related event handlers** JavaScript support some keyboard event handlers, which are very useful in responding to the user input when keys on a keyboard are pressed. These events are what make it possible for JavaScript to “react” when a key is pressed:

Table: Keyboard related event handlers

Event handler	functions
Onkeypress	invokes JavaScript code when a key is pressed
Onkeydown	invokes JavaScript code when a key is held down (but not yet released)
onkeyup	invokes JavaScript code when a key is has been released after being pressed.

These events can be bind to most elements on a game page. But, you will probably stick to either the “document” element in most cases. For example, the code below uses the **onkeypress** property of the **document** element to call the **show** function, which in turns changes the display attribute’s value from “**none**” to “**inline**”. In other words, when the user presses any key, the **cat.gif** appears.

```
<script type="text/javascript">
function show() {
m1.style.display="inline";
}
document.onkeypress=show;
</script>


```

In the following example, the **onKeyDown** is used to display the time the page was loaded, while **onKeyUp** displays the current time.

```
<script>
function init_time() {
init_time = Date();
}
</script>

<body onLoad=init_time()
onKeyDown="t1.innerText=init_time"
onKeyUp="t2.innerText=Date()">
Starting time: <b id=t1></b><br>
Ending time: <b id=t2></b>
```

A sample output looks:

Starting time: **Sat Feb 24 21:39:17 2007**  
Ending time: **Sat Feb 24 21:41:39 2007**

To further demonstrate how the keyboard-related event handlers work in game programming, create a new project called kb.htm with the following contents:

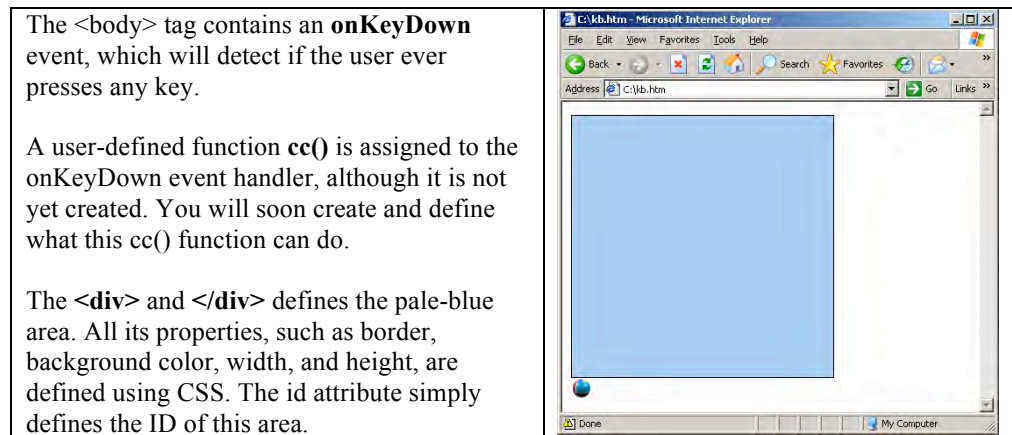
```
<html>
<body onkeydown=cc()>

    <div id=area1 style="border:solid 1 black;
                        background-color:#abcdef;
                        width:300; height:300;">

    </div>
    

</body>
</html>
```

Use Internet Explorer to run the code. It creates a pale-blue area with a rolling ball below it.



The <img> tag loads the animated gif file. It also uses CSS to define properties. Noticeably, the position is set to be absolute, which means the position value set (**top, left**) is based on the Web browser's origin. The ID of the gif file is ball.

#### The KeyCode

The **keyCode** property of the Event object returns UniCode value of key pressed. The returned value is the key's numerical value, not the American National Standards Institute (ANSI) value. You can use keyCode to detect when the user has pressed an arrow or function key, which cannot be specified by the key property.

The syntax is:

```
event.keyCode
```

For example, you can display each key's key code on the status bar by using the following code:

```
<body onKeyDowN="status=event.keyCode">
```

Press the Alt key, you see a value 18 on the status bar:

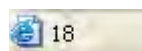


Table: Common Key Codes

Key	Code	Key	Code	Key	Code	Key	Code
A	65	0	48	+	107	[Alt]	18
B	66	1	49	-	109	[Enter]	13
C	67	2	50	*	106	[Shift]	16
D	68	3	51	/	111	[Ctrl]	17
E	69	4	52	`	192	[CapsLock]	20
F	70	5	53	,	188	[Esc]	27
G	71	6	54	.	190	[Backspace]	8
H	72	7	55	/	191	[Insert]	45
I	73	8	56	;	186	[Home]	36
J	74	9	57	\	222	[PageUp]	33
K	75	F1	112	[	219	[Delete]	46
L	76	F2	113	]	221	[End]	35
M	77	F3	114	\	220	[PageDown]	34
N	78	F4	115	=	187	[PrintScreen]	
O	79	F5	116			[ScrollLock]	145
P	80	F6	117			[Pause]	19
Q	81	F7	118				
R	82	F8	119				
S	83	F9	120				
T	84	F10	121				
U	85	F11	122				
V	86	F12	123				
W	87	←	37				
X	88	↑	38				
Y	89	→	39				
Z	90	↓	40				

You can create a function **cc()** and use the `keyCode` property of the event object to determine whether or not the user presses the **Left** arrow key (←). This Left arrow key has a numerical value 37. A simple **if..then** decisive logic can make this decision.

```
<html>
<script>
function cc() {
  if(event.keyCode==37) {
    ball.style.pixelLeft -= 10;
  }
}
</script>

<body onkeydown=cc()>
  <div id=area1 style="border:solid 1 black;
    background-color:#abcdef;
    width:300; height:300;">

  </div>
</body>
</html>
```

In the following line, “ball” is the ID of the gif file, so “ball.style.left” represents the horizontal value of the position value set (**top, left**). “**style**” is a keyword that indicates Cascading Style Sheet.

```
ball.style.pixelLeft -= 10;
```

The original value of “ball.style.left” is set to the 10, as specified by “left:10”. The above line is technically the same as:

```
ball.style.pixelLeft = ball.style.pixelLeft - 10;
```

In the old days, the above line would look:

```
ball.style.left = eval(ball.style.left.replace('px','')) - 10;
```

The difference between **left** and **pixelLeft** is that the value of left is a string, such as **120px**. The value of pixelLeft is an integer, such as **120**. Since a string value cannot be calculated arithematically, you need to use the **replace()** method to remove the ‘**px**’ substring.

```
ball.style.left.replace('px','')
```

Consequently, a value of 120px becomes 120. But, 120 is still a string made of 1, 2, and 0, not an integer.

The **eval()** method then convert the number-like string to the **integer** data type. In other words, 120 now means one-hundred-and-twenty, no longer a string that reads one-two-zero. With the new XHTML, CSS and JavaScript standard, you can ignore this old technique. However, the instructor purposely brings up this issue for your reference (see learning activity #1 for details).

The following line tells the computer to subtract 10 from the current “ball.style.pixelLeft” each time when the user press the **Left** arrow key (←).

```
if(event.keyCode==37) {
    ball.style.pixelLeft -= 10;
}
```

According to the following table, the Up, Right, and Down arrow keys have values of 38, 39, and 40. Similarly, you can use the following line to add 10 to the current “ball.style.left” each time when the user press the **Right** arrow key (→).

```
if(event.keyCode==39) {
    ball.style.pixelLeft += 10;
}
```

The change the vertical position, you need to modify the current value of “ball.style.top”, which represents the vertical value of the position value set (**top, left**).

```
if(event.keyCode==38) {
    ball.style.pixelTop -= 10;
}

if(event.keyCode==40) {
    ball.style.pixelTop += 10;
}
```

The code now looks:

```
<html>
<script>
function cc() {
    if(event.keyCode==37) {
        ball.style.pixelLeft -= 10;
    }

    if(event.keyCode==39) {
        ball.style.pixelLeft += 10;
    }
}
```

```

if(event.keyCode==38) {
    ball.style.pixelTop -= 10;
}

if(event.keyCode==40) {
    ball.style.pixelTop += 10;
}

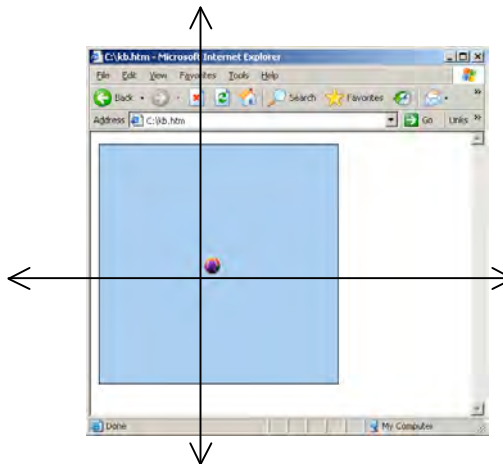
}
</script>

<body onkeydown=cc()>
    <div id=area1 style="border:solid 1 black;
        background-color:#abcdef;
        width:300; height:300;">

        
    </div>
</body>
</html>

```

Use Internet to run the code, and use the ←, ↑, →, and ↓ keys to move the ball.



By the way, some keys, like [Shift], [Control] and [Alt], aren't normally thought of as sending characters, but modify the characters sent by other keys. For many keys, no key codes are returned on keydown and keyup events. Instead the keyCode value is just zero. Characters that give a zero keycode when typed include those listed below, as well as any key when the Alt/Option key is held down.

- \_ ~ ! @ # \$ % ^ & \* ( ) + | : < > ?

Internet Explorer provides the following properties.

Table: Keyboard related properties of the Event object

Properties	Description
altKey, ctrlKey, shiftKey	Boolean properties that indicate whether the Alt, Ctrl, Meta, and Shift keys were pressed at time of the event.
keyCode	Property indicating the Unicode for the key pressed.
type	A string indicating the type of event, such as "mouseover", "click", etc.

For example, to create a code that requires the user to hold the Shift key and then press the C key to play the C# sound, use:

```
<script>
function cs() {
  if ((event.shiftKey) && (event.keyCode==67)) {
    document.Cs.play();
  }
}
</script>
<body onKeyDown=cs()>
<embed src="Cs0.wav" autostart=false hidden=true name="Cs"
mastersound>
</body>
```

There are few properties that work only with Netscape or Firefox, but are not discussed in this lecture. They are:

Properties	Description
metaKey	Boolean property that indicate whether the Meta key was pressed at the time of the event.
charCode	Property indicating the Unicode for the key pressed.
which	Legacy property indicating the Unicode for the key pressed.

Both *if..then* and *switch..case* statements are frequently used to make decision as what to respond to the user's input from keyboard. For example,

```
<script>
function move_witch() {

  if (event.keyCode==37) {
    w1.style.pixelLeft -= 10;}
  if (event.keyCode==38) {
    w1.style. pixelTop -= 10;}
  if (event.keyCode==39) {
    w1.style. pixelLeft += 10;}
  if (event.keyCode==40) {
    w1.style. pixelTop += 10;}
}
</script>
```

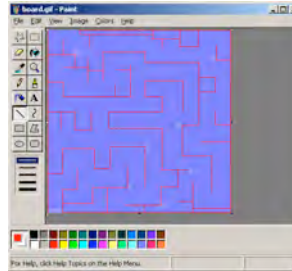
You can rewrite the above code using **switch..case** statement.

```
<script>
function move_witch() {
var i = event.keyCode;
switch (i) {

case 37:
w1.style.pixelLeft -= 10; break;
case 38:
w1.style. pixelTop -= 10; break;
case 39:
w1.style. pixelLeft += 10; break;
case 40:
w1.style. pixelTop += 10; break;
}
}
</script>
```

Technically speaking, the **switch..case** statement is a multi-way decision statement. Unlike the multiple-decision statement that can be created using **if..then**. The switch statement evaluates the conditional expression and tests it against numerous constant values. The branch corresponding to the value that the expression matches is taken during execution.

Sample Games      Use Paint to create a gif file (named board.gif) that looks:



Modify the kb.htm file to:

```
<html>
<script>
function cc() {
  if(event.keyCode==37) {
    ball.style.left -= 10;
  }

  if(event.keyCode==39) {
    ball.style.left += 10;
  }

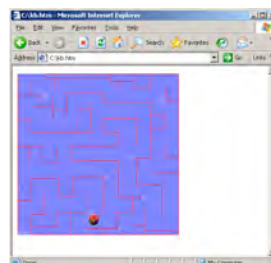
  if(event.keyCode==38) {
    ball.style.top -= 10;
  }

  if(event.keyCode==40) {
    ball.style.top += 10;
  }
}
</script>

<body onkeydown=cc()>
  
  

</body>
</html>
```

Use Internet to run the code, you just create a maze game.



This code can be written using the switch..case statement. For example,

```
<html>
<script>
function cc() {
var i = event.keyCode;
switch (i) {
case 37:
ball.style.pixelLeft -= 10;
break;

case 39:
ball.style.pixelLeft += 10;
break;

case 38:
ball.style.pixelTop -= 10;
break;

case 40:
ball.style.pixelTop += 10;
break;
}
}
</script>

<body onkeydown=cc()>



</body>
</html>
```

Use keyboard to control the width and height of an object

Many objects have the properties of width and height. You can increase or decrease the values of them to create visual effects. In the following example, the <img> tag is given and ID m1. Consequently, “m1.style.width” represents the **width** property of this image object.

```
<html>
<script>
function cc() {
if (event.keyCode==38) {
m1.style.pixelWidth += 10;
}

if (event.keyCode==40) {
m1.style.pixelWidth -= 10;
}
}
</script>
<body onKeyUp=cc()>
<center></center>
</body>

</html>
```

When you run the code, the Up and Down arrow keys will increase and decrease the value of width. On the screen, the change of value creates effect of zoom in and zooms out.



## Review Questions

1. Which event handler invokes JavaScript code when a key is held down, but not yet released?
  - A. onKeyUp
  - B. Onkeypress
  - C. Onkeydown
  - D. onkeyup

2. Which is not a keyboard related event handlers?
  - A. onKeyUp
  - B. Onkeypress
  - C. Onkeydown
  - D. onkeyup

4. Given the following code block, which statement is correct?

```
<body onKeyDown="status = new Date();"
  onLoad="document.title=new Date()"
  onKeyUp="p1.innerText=new Date()">
```

- A. When the user loads the page, the current data and time is display on the status bar.
- B. When the user press and hold the [Enter] key, the current data and time is displayed on the status bar.
- C. When the user press and then release the [Enter] key, the current data and time is displayed on the status bar.
- D. All of the above

4. Which property of the Event object returns UniCode value of key pressed?
  - A. event.keyType
  - B. event.keyCode
  - C. event.key
  - D. event.keypress

5. Given the following code block, which statement is incorrect?

```
<body onKeyDown="status=event.keyCode">
```

- A. When you press any key, the status bar displays the string "event.keyCode".
- B. When you press the Alt key, the status bar displays the value 18.
- C. When you press the left arrow key, the status bar displays the value 37.
- D. When you press the P key, the status bar displays the value 80.

6. When Unicode value is what you get when you press the Scroll Lock key?
  - A. 145
  - B. 125
  - C. 135
  - D. 105

7. Given the following code, which statement is correct?

```
ball.style.left.replace('px','')
```

- A. It uses the replace() method to remove "px" from "10px".
- B. It uses the replace() method to remove "10px" from "10px".
- C. It sets the value of ball.style.left to null.
- D. All of the above

8. Which property of the event object returns a boolean outcome?

- A. altKey
- B. ctrlKey
- C. shiftKey
- D. All of the above

9. Given the following code block, which statement is correct?

```
if ((event.shiftKey) && (event.keyCode==37)) {  
    m1.src="1.gif";  
}
```

- A. To change the value of src property of m1 object to "1.gif", you can press either Shift key or C key.
- B. To change the value of src property of m1 object to "1.gif", you can press both Shift key and C key.
- C. To change the value of src property of m1 object to "1.gif", you cannot press Shift key nor C key.
- D. None of the above.

10. Which is the correct way to detect what key has been pressed?

- A. <body onKeyDown="status=event.keyCode">
- B. <body onKeyDown="status=event.keyPressCode">
- C. <body onKeyDown="status=event.keyTypeCode">
- D. <body onKeyDown="status=event.keyOnCode">

## Game Programming

### Lab #6

#### Preparation #1:

1. Create a new directory named **C:\games**.
2. Use Internet Explorer to go to **<http://business.cypresscollege.edu/~pwu/cis261/download.htm>** to download lab6.zip (a zipped) file. Extract the files to C:\games directory.

#### Learning Activity #1: Labyrinth

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named **C:\games\lab6\_1.htm** with the following contents:

```
<html>
<script>
function cc() {
var i = event.keyCode;
  switch (i) {
    case 37:
      ball.style.left=eval(ball.style.left.replace('px','')) - 10;
      break;

    case 39:
      ball.style.left=eval(ball.style.left.replace('px','')) + 10;
      break;





    case 38:
      ball.style.top=eval(ball.style.top.replace('px','')) - 10;
      break;

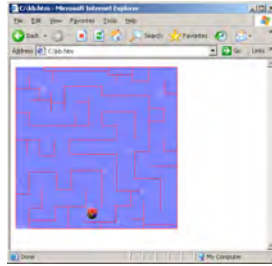
    case 40:
      ball.style.top=eval(ball.style.top.replace('px','')) + 10;
      break;
  }
}
</script>

<body onkeydown=cc()>
  
  

</body>
</html>
```

Note: This version use *switch..case*, so be sure to compare this version with the one in the lecture which use *if..then* statement.

3. Test the program. Use , , , and  arrow keys to move the ball. A sample output looks:



## Learning Activity #2: Music Keyboard

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to open the C:\games\lab6\_2.htm (you just downloaded it) which has the contents inside the dash lines:

```
<html>
<style>
.wKey {
    position:absolute;
    top:20;
    background-color:white;
    border-top:solid 1 #cdcdcd;
    border-left:solid 1 #cdcdcd;
    border-bottom:solid 4 #cdcdcd;
    border-right:solid 1 black;
    height:150px;
    width:40px
}

.bKey {
    position:absolute;
    top:20;
    background-color:black;
    border:solid 1 white;
    height:70px;
    width:36px
}
</style>

<script>
<!-- for middle C -->
function CDown() {
    midC.style.borderTop="solid 1 black";
    midC.style.borderLeft="solid 1 black";
    midC.style.borderBottom="solid 1 black";
    midC.style.borderRight="solid 1 #cdcdcd";
}

function CUp() {
    midC.style.borderTop="solid 1 #cdcdcd";
    midC.style.borderLeft="solid 1 #cdcdcd";
    midC.style.borderBottom="solid 4 #cdcdcd";
    midC.style.borderRight="solid 1 black";
}

<!-- for middle D -->
```

```

function DDown() {
midD.style.borderTop="solid 1 black";
midD.style.borderLeft="solid 1 black";
midD.style.borderBottom="solid 1 black";
midD.style.borderRight="solid 1 #cdcdcd";
}

function DUp() {
midD.style.borderTop="solid 1 #cdcdcd";
midD.style.borderLeft="solid 1 #cdcdcd";
midD.style.borderBottom="solid 4 #cdcdcd";
midD.style.borderRight="solid 1 black";
}

<!-- for middle E -->
function EDown() {
midE.style.borderTop="solid 1 black";
midE.style.borderLeft="solid 1 black";
midE.style.borderBottom="solid 1 black";
midE.style.borderRight="solid 1 #cdcdcd";
}

function EUp() {
midE.style.borderTop="solid 1 #cdcdcd";
midE.style.borderLeft="solid 1 #cdcdcd";
midE.style.borderBottom="solid 4 #cdcdcd";
midE.style.borderRight="solid 1 black";
}

```

3. Add the following lines to the existing contents:

```

function Downkeys() {
var i = event.keyCode;

if (event.shiftKey) {
switch (i) {
case 67: document.Cs.play(); break;
case 68: document.Ds.play(); break;
}
}

else {
switch (i) {
case 67: CDown(); document.Csound.play(); break;
case 68: DDown(); document.Dsound.play(); break;
case 69: EDown(); document.Esound.play(); break;
}
}
}

function Upkeys() {
var i = event.keyCode;
switch (i) {
case 67: CUp(); break;
case 68: DUp(); break;
case 69: EUp(); break;
}
}
document.onkeydown=Downkeys;
document.onkeyup=Upkeys;
</script>

```

```

<div>

<span class="wKey" id="midC" style="left:50"></span>
<span class="wKey" id="midD" style="left:91"></span>
<span class="wKey" id="midE" style="left:132"></span>

<span class="bKey" id="cSharp" style="left:72"></span>
<span class="bKey" id="dSharp" style="left:114"></span>
</div>

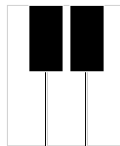
<!-- white key sound files -->
<embed src="C0.wav" autostart=false hidden=true name="Csound" mastersound>
<embed src="D0.wav" autostart=false hidden=true name="Dsound" mastersound>
<embed src="E0.wav" autostart=false hidden=true name="Esound" mastersound>

<!-- black key sound files -->
<embed src="Cs0.wav" autostart=false hidden=true name="Cs" mastersound>
<embed src="Ds0.wav" autostart=false hidden=true name="Ds" mastersound>

</html>

```

4. Test the program. Press C, D, and Key first, and then Shift+C, Shift+D to play corresponding sounds. A sample output looks:



### Learning Activity #3: snake game

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab6\_3.htm with the following contents:

```

<html>
<script>
var x=190; y=384;
var k=1; n=0;
function draw(){
var i = event.keyCode;
switch (i) {
case 37: n=1; k=0; break;
case 38: n=0; k=1; break;
case 39: n=-1; k=0; break;
case 40: n=0; k=-1; break;
}
}

function cc() {

if ((x<=10) || (x>=394) || (y<=-10) || (y>=400)) {
p1.innerText="Game over!";
}
else {
y-=k; x-=n;
codes = "<span style='position:absolute;left:"+x;

```

```





codes += "; top:" + y + "'>.</span>";
areal.innerHTML += codes;
setTimeout("cc()", 0.1);
}
}

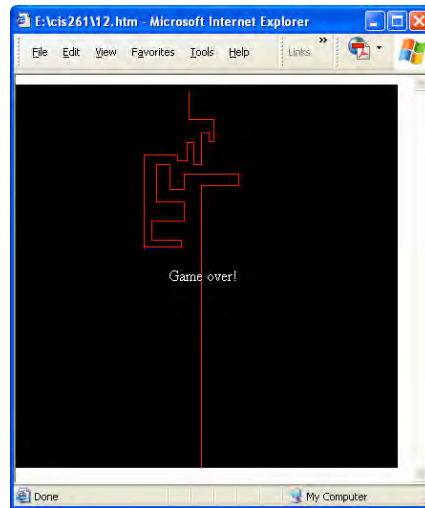
</script>

<body OnLoad=cc() onKeyDown=draw()>
<div id=areal style="position:absolute; width:400;
  height:400; border:solid 3 black; color:red;
  background-color: black; left:0; top:10;"></div>

<p id=p1 style="position:absolute; left:160;top:200; color: white"></p>
</body>
</html>

```

3. Test the program. Use , , , and  arrow keys to move. A sample output looks:



#### Learning Activity #4: Flying witch

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab6\_4.htm with the following contents:

```

<html>

<style>
img { position:absolute}
</style>

<script>
function init() {
a1_x=document.body.clientWidth;
a1_y=Math.floor(Math.random()*document.body.clientHeight);

c1_x=0;
c1_y=Math.floor(Math.random()*document.body.clientHeight);

```

```

w1_x=200;
w1_y=Math.floor(Math.random()*document.body.clientHeight);

b1_x=Math.floor(Math.random()*document.body.clientWidth);
b1_y=document.body.clientHeight;

codes="<img id=a1 src=airplane.gif style='left:"+a1_x+";top:"+a1_y+"'>";
codes+="<img id=c1 src=cloud.gif style='left:"+c1_x+";top:"+c1_y+"'>";
codes+="<img id=w1 src=witch.gif style='left:"+w1_x+";top:"+w1_y+"'>";
codes+="<img id=b1 src=bird.gif style='left:"+b1_x+";top:"+b1_y+"'>";
p1.innerHTML=codes;

airplane();
cloud();
witch();
bird();
}

function airplane() {

    if (a1.style.pixelLeft > 0) {
        a1.style.pixelLeft-=10;
    }
    else {
        a1.style.left=document.body.clientWidth;
        a1.style.top=Math.floor(Math.random()*document.body.clientHeight);
    }
    setTimeout("airplane()", 20);
}

function cloud() {

    if (c1.style.pixelLeft < document.body.clientWidth) {
        c1.style.pixelLeft += 10;
    }
    else {
        c1.style.pixelLeft=0;
        c1.style.pixelTop=Math.floor(Math.random()*document.body.clientHeight);
    }
    setTimeout("cloud()", 100);
}

function witch() {

    if (w1.style.pixelLeft < document.body.clientWidth) {
        w1.style.pixelLeft += 10;
    }
    else {
        w1.style.pixelLeft=0;
        w1.style.pixelTop=Math.floor(Math.random()*document.body.clientHeight);
    }
    setTimeout("witch()", 100);
}

function bird() {

    if (b1.style.pixelTop > 0) {
        b1.style.pixelTop -= 10;
        b1.style.pixelLeft -= 10;
    }
    else {
        b1.style.pixelLeft=Math.floor(Math.random()*document.body.clientWidth);
        b1.style.pixelTop=document.body.clientHeight;
    }
}

```

```





    }
    setTimeout("bird()", 50);
}

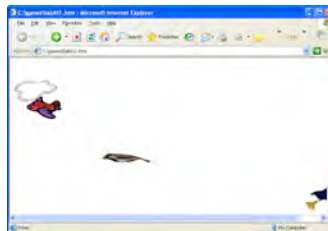
function move_witch() {
    var i = event.keyCode;
    switch (i) {

        case 37:
            w1.style.pixelLeft -= 10; break;
        case 38:
            w1.style.pixelTop -= 10; break;
        case 39:
            w1.style.pixelLeft += 10; break;
        case 40:
            w1.style.pixelTop += 10; break;
    }
}
</script>

<body onLoad=init() onKeyDown="move_witch()">
<span id=p1></span>
</body>
</html>

```

- Test the program. Use , , , and  arrow keys to move the witch to avoid crashing into any object. A sample output looks:



### Learning Activity #5: Jet Fighters

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

- Change to the C:\games directory.
- Use Notepad to create a new file named **C:\games\lab6\_5.htm** with the following contents:

```

<html>
<script>

function init() {
    ap01.style.pixelLeft=Math.floor(Math.random()*300);
    cc();
}

var k = 2;

function cc() {
    if (ap01.style.pixelLeft > document.body.clientWidth - 20) { k = -2; }
    else if (ap01.style.pixelLeft < 10) { k = 2; }
    ap01.style.pixelLeft += k;
    setTimeout("cc()",5);
}

```

```

}

function fly() {
    if(event.keyCode==37) {
        st01.style.pixelLeft -= 10;
    }

    if(event.keyCode==39) {
        st01.style.pixelLeft += 10;
    }

    if(event.keyCode==38) {
        st01.style.pixelTop -= 10;
    }

    if(event.keyCode==40) {
        st01.style.pixelTop += 10;
    }

    if(event.keyCode==83) {
        b01.style.pixelLeft = st01.style.pixelLeft + 20;
        status = st01.style.pixelWidth;
        b01.style.pixelTop = st01.style.pixelTop;
        b01.style.display='inline';
        fire();
    }
}

function fire() {
    b01.style.pixelTop -= 10;
    setTimeout("fire()",50);
}

</script>
<body id="bd" onkeydown=fly() onLoad=init()>









<span id=b01 style="position:absolute;display:none">!</span>

</body>

</html>

```

Note: Because collision detection and response is the topic of a later lecture, how you can modify this code so that the bullet can destroy the plane will be discussed in a later lecture.

3. Test the program. Use , , , and  arrow keys to move airplane. Press the S key to shoot a bullet. A sample output looks:



**Submittal**

Upon completing all the learning activities,

1. Upload all files you created in this lab to your remote web server.
2. Log in to Blackboard, launch Assignment 06, and then scroll down to question 11.
3. Copy and paste the URLs to the textbox. For example,
  - [http://www.geocities.com/cis261/lab6\\_1.htm](http://www.geocities.com/cis261/lab6_1.htm)
  - [http://www.geocities.com/cis261/lab6\\_2.htm](http://www.geocities.com/cis261/lab6_2.htm)
  - [http://www.geocities.com/cis261/lab6\\_3.htm](http://www.geocities.com/cis261/lab6_3.htm)
  - [http://www.geocities.com/cis261/lab6\\_4.htm](http://www.geocities.com/cis261/lab6_4.htm)
  - [http://www.geocities.com/cis261/lab6\\_5.htm](http://www.geocities.com/cis261/lab6_5.htm)

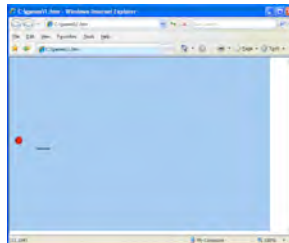
No credit is given to broken link(s).

## Game Programming

### Lecture #7 Handling moving Objects

**Introduction** Moving objects is an extremely important topic in all games because they provide an effective means of conveying movement while also allowing objects to interact with one another. By learning how to move objects in a game, you can create some interesting games.

For example, in the ball bouncing game (you created in a previous lecture), there are two objects: the ball and the paddle (a customized cursor). These two objects must be coded to be movable because they all move and interact with each other. The ball floats inside a given area while the paddle can be moved by the player to bounce back the ball.



In this example, the movement of ball is in essence the movement of an image file, while the paddle is the mouse cursor (whose movement is handled by the operating systems). But it points out the truth that moving objects are typically bitmap images, and the game programmers' job is to keep track of the position, velocity, width, height, Z-index (or layers), and visibility. A previous lecture had explained how the coordinate system is used by the computer to position an object using (x, y) format.

In terms of DHTML games, CSS is responsible for the appearance of objects, while a scripting language (e.g. Javascript) is responsible for how objects move, behave, and interact with other objects. For example, in the ball bouncing game,

- CSS is used to define the dimension (width and height), background color, and border of the area, in which the ball is defined by CSS its initial position and visibility.
- JavaScript codes creates two functions **dd()** and **ff()**. The dd() function moves the ball continuously from top-left to right-bottom direction. The ff() function is a reversed function of dd().

```
<html><body onClick="t1.style.display='block';dd()">

<script>
var j=0;
var i=Math.round(Math.random()*1000);

function dd() {

t1.style.left=i;
t1.style.top=j;
status=" (" +i+", "+j+" ) ";

i++;j+=Math.round(Math.random()*10);

if ((j>=400) || (i>=600)) {clearTimeout(s1);ff()}
if (i<=8) {i=8;i+=20;j+=10}
s1=setTimeout("dd()", 50);
}
```

```

function ff() {
t1.style.left=i;
t1.style.top=j;
status="("+i+", "+j+") ";
j-=10;i-=5;
if (j<=0) {clearTimeout(s2);dd();} else
{s2=setTimeout("ff()",50);}
}

</script>
<div style="position:absolute; top:0 ;left:0; width:608;
height:408; background-color:#abcdef;
cursor:vertical-text">


</div>

</body></html>

```

The movement of the ball is determined by its  $(x, y)$  coordinates in the browser's body area. DHTML uses `left` and `top` attributes to represent  $x$  and  $y$  respectively. The syntax to use them are:

```
objectID.style.left
```

and

```
objectID.style.top
```

where **objectID** is the ID you assigned to the object you want to move. For example, `t1` is the ID of the `ball.gif` according to the following statement. The object's ID varies according to the object, while the keywords **style**, **left**, and **top** must stay.

```

```

When the values of an object's  $(x, y)$  change, the object moves from its original location to the new location. In the following example, two variables *i* and *j* will assign values of  $(x, y)$  to the **t1** object.

```

function dd() {

t1.style.left=i;
t1.style.top=j;
.....

i++;j+=Math.round(Math.random()*10);
.....
s1=setTimeout("dd()",50);
}

```

The **setTimeout()** method will call the `dd()` function every 50 milliseconds, and each time when `dd()` executes, the following line forces *i* and *j* to change their values. In other words, the  $(x, y)$  value of **t1** is renewed each time when `dd()` executes. Since **t1** receives a new position value set  $(x, y)$ , it keeps on moving from previous  $(x, y)$  to current  $(x, y)$  to create a visual effect of moving ball.

Creating a  
motion effects

A motion object must have identifiable properties with valid values for the computer to know how to manage its motion. Game programming using object-oriented technique to handle motion is basically the implementation of an object's properties. Most programming languages provide the following properties to define the motion of a given object in a game that is capable of

moving over time. The basic techniques for finding an object's property are the same in very language. In terms of DHTML, for example, the general syntax for finding an element's HTML property values is:

```
getElementById(objectID).style.PropertyName;
```

where, **objectID** is the ID you assign to a given object, while the word “**style**” is a keyword that must stay as it is. As to the **PropertyName**, it must be a valid property. In DHTML game programming, you will frequently use the following properties:

- position (such as left, top, pixelLeft, pixelTop, etc.)
- visibility
- z-index

Consider the following example; **m1** is the ID of the image file g1.gif (which is an object).

```
<html>
<script>
function init()
{
  var x=document.getElementById('m1').style.pixelLeft;
  var y=document.getElementById('m1').style.pixelTop;
  status="("+x+", "+y+")";
}
</script>

<body onLoad=init()>
<img id=m1 src=g1.gif>
</body>
</html>
```

The following line tells the computer to get the value of pixelLeft of the **m1** object.

```
var x=document.getElementById('m1').style.pixelLeft;
```

In previous lectures, you had learned that the above line can be simplified to:

```
var x=m1.style.pixelLeft;
```

So, what is the point using **getElementById()**? When using it with **srcElement**, you will make your long codes very concise. The **srcElement** property sets or retrieves the object that fired the event. The syntax is:

```
[ oObject = ] event.srcElement
```

The optional [ **oObject** = ] part refers to the object that specifies or receives the event that fired. For example,

```
<html>
<script>
function change_color() {
var oldID = event.srcElement.id;
document.getElementById(oldID).style.color=oldID;
}
</script>

<button id="red" onclick="change_color()">Red</button>
<button id="blue" onclick="change_color()">Color</button>
</html>
```

In the above code, the following line detects the ID of the button you click (e.g. “red” or “green”),

and assign the value to a variable named **oldID**.

```
var oldID = event.srcElement.id;
```

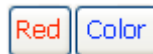
The following line tells the computer to change the text color of the button to the value held by the oldID variable (which is either “red” or “green”).

```
document.getElementById(oldID).style.color=oldID;
```

You will be using this programming technique often in the later lectures. When executing the above code, you first see:



After clicking on the buttons, the colors change from black to red and black to blue.



The SrcElement DOM object is generated any time that an event is called, and it contains the element that called the event in the first place. One of the principle advantages of using it is that you can radically simplify your DHTML scripting code by assigning an event on a single container object, then checking the SrcElement to see which object within that container actually fired the event.

#### DHTML position systems

DHTML uses the **position** property to sets or retrieves the type of positioning used for the object. The syntax is:

```
objectID.style.position=value
```

Possible values are:

- **static**: Default. Object has no special positioning; it follows the layout rules of HTML.
- **absolute**: Object is positioned relative to parent element’s position or to the body object if its parent element is not positioned using the **top** and **left** properties.
- **relative**: Object is positioned according to the normal flow, and then offset by the **top** and **left** properties.

A simple demonstration illustrates the point. Suppose that you had three <DIV> elements within another <DIV> element, one of each color red, green or blue. You can place a general event handler on the containing DIV that can then be used to intercept either the outside division or any of the three smaller DIV elements.

```
<script>
function getColor(){
  var src=event.srcElement;
  alert(src.style.backgroundColor);
}
</script>

<div
style="background-color:black; color:white; width:500px;
cursor:hand;"
onclick="getColor()">Black

<div
style="background-color:red; color:black; width:150px;
```

```

position:relative">
Red Box</div>

<div style="background-color:green; color:black; width:150px;
position:relative">
Green Box</div>

<div
style="background-color:blue; color:black; width:150px;
position:relative">
Blue Box</div>

</div>

```

This technique is common for dealing with a large number of items that have similar actions and are contained in a single element. Collapsible trees, subordinate selections, input elements, and behaviors can all make use of this technique.

Setting the property to absolute pulls the object out of the "flow" of the document and positions it regardless of the layout of surrounding objects. If other objects already occupy the given position, they do not affect the positioned object, nor does the positioned object affect them. Instead, all objects are drawn at the same place, causing the objects to overlap. This overlap is controlled by using the z-index attribute or property. Absolutely positioned objects do not have margins, but they do have borders and padding.

To enable absolute positioning on an object you must specify at least one of the top, bottom, left, or right properties, in addition to setting the position property to absolute. Otherwise, these positioning properties use their default value of absolute, which causes the object to render immediately after the preceding elements, according to the layout rules of HTML.

#### Object Visibility and Display

The **visibility** property sets if an element should be visible or invisible. The **display** property sets how/if an element is displayed. Invisible elements take up space on the page. Use the "display" property to create invisible elements that do not take up space.

The syntax is:

```
objectID.style.visibility=value
```

and

```
objectID.style.display=value
```

Command values of Visibility project include:

Value	Description
visible	The element is visible
hidden	The element is invisible
collapse	When used in table elements, this value removes a row or column, but it does not affect the table layout. The space taken up by the row or column will be available for other content. If this value is used on other elements, it renders as "hidden"

Command values of Display project include:

Value	Description
none	The element will not be displayed
block	The element will be displayed as a block-level element, with a line break before and after the element
inline	The element will be displayed as an inline element, with no line break before or after the element

list-item	The element will be displayed as a list
run-in	The element will be displayed as block-level or inline element depending on context
compact	The element will be displayed as block-level or inline element depending on context

For example,

```
<html>
<u onClick=this.style.display="none">Make me disappear!</u>
This is a line.<br>
<b onClick=this.style.visibility="hidden">Make me invisible!</b>
This is another line.
</html>
```

When click the sentence “Make me disappear!”, it disappears (being removed). When click the sentence “Make me invisible!”, it becomes invisible (their character space remain there)

Layering  
Objects – z-  
index property

Most of the Web pages are meant to be 2D (2 dimensional), so objects on the Web pages can be positioned by X and Y coordinates, or its horizontal and vertical positioning. CSS positioning uses the Z axis to layer objects on top of each other. With the z-index property, you can specify the layer on which an object lies.

By setting the z-index higher or lower, you can move the object up or down in the stack. Basically, the bigger the value is; the more front its layer will be. Also,

- A positive value positions the element above text that has no defined z-index.
- A negative value positions it below.
- Set this parameter to null to remove the attribute.

The syntax is:

```
objectID.style.zIndex="value";
```

Z-index property is relative to x and y-axis, it will not work without x and y-axis, that is, top and left properties. For example,

```
<html>
<style>
u {z-index:3;position:absolute;left:55;top:25;background-
color:red}
b {z-index:2;position:absolute;left:65;top:35;background-
color:green}
i {z-index:1;position:absolute;left:75;top:45;background-
color:blue}
</style>

<u>This is layer1.</u>
<b>this is layer2.</b>
<i>This is layer3.</i>
</html>
```

One can also integrate the CSS style with HTML tags, for instance:

```
<html>
<style>
u {z-index:3;position:absolute;top:55px;left:25px;background-
color:red}
i {z-index:2;position:absolute;top:65px;left:55px;background-
color:green}
```

```

b {z-index:1;position:absolute;top:75px;background-color:blue}
</style>
<u>this goes to the top.</u>
<i>this goes to the middle.</i>
<b>this goes the bottom.</b>
</html>

```

The output looks:



#### Continuous motion

A motion object also have built-in functions, such as start, stop, turn left, turn right, and so on. The term "built-in" implies that the game programmer must create these functions and given them to the object. Most programming languages provide methods to allow game programmers to create "user-defined function", which then serves as the "built-in" function of the object for the players to use to control the object's motion.

JavaScript is relatively weak on the support of such methods, but you can use the following JavaScript method to handle motions of your DHTML games.

- **setTimeout()** - executes a code some time in the future
- **clearTimeout()** - cancels the setTimeout()

Note: The setTimeout() and clearTimeout() are both methods of the HTML DOM Window object.

To use the setTimeout() method, the formal syntax is:

```
var variableName=setTimeout("statement", milliseconds)
```

Although, you can simplify the syntax to:

```
setTimeout("statement", milliseconds);
```

it is highly recommended that you use the formal syntax, so you can use the clearTimeout() method to cancel it. The syntax of clearTimeout() method is:

```
clearTimeout(variableName)
```

Due to the object-oriented programming model, methods are controlled by the programming language's run-time environment (meaning the background engine), so they cannot be destroyed, deleted, or terminated by the user.

You can create an object, such as a variable in this setTimeout() case, to transfer all the setTimeout() method's function to this object. Consequently, you can destroy, delete, or terminate the object any time you want by using clearTimeout(variableName).

Additionally, the setTimeout() method returns a value - In the statement above, the value is stored in a variable called t. If you want to cancel this setTimeout(), you can refer to it using the variable name.

#### Drag and drop

"Drag and drop" describes a particular action you can move an element from one location to another by dragging it with the mouse. Click an object, such as a image, then hold down the mouse button as you drag the object to a new location. You drop the object by releasing the mouse button.

This function is made possible by a good handling of (x, y) coordinates using mouse buttons'

positioning systems: **event.clientX** and **event.clientY**. Given the following code which is saved as an external code file (with a name such as dragndrop.js). You can use this code file as library code to provide drag-and-drop function to other DHTML codes.

```
var dragapproved=false
var z,x,y

function move(){
  if (event.button==1 && dragapproved){
    z.style.pixelLeft=temp1+event.clientX-x
    z.style.pixelTop=temp2+event.clientY-y
    return false
  }
}

function drags(){
  if (!document.all)
    return
  if (event.srcElement.className=="drag"){
    dragapproved=true
    z=event.srcElement
    temp1=z.style.pixelLeft
    temp2=z.style.pixelTop
    x=event.clientX
    y=event.clientY
    document.onmousemove=move
  }
}
document.onmousedown=drags
document.onmouseup=new Function("dragapproved=false")
```

There are four user-defined variables: dragapproved, z, x, and y. The variable **dragapproved** is merely used as a reference that indicates whether or not any mouse button is currently pressed and held. If no mouse button is pressed and held, the value of dragapproved is **false**.

The variable **z** is just a temporarily name of the object that is being dragged. It is used only when an object is dragged.

The variables **x** and **y** represents the mouse cursor's (x, y) coordinates, which is exactly the **event.clientX** and **event.clientY**. The values of x and y variable change as the mouse cursor moves.

The following statement triggers the drags() function when an object is dragged.

```
document.onmousedown=drags
```

In the drags() function, the following line verifies if an object being dragged has a class name "drag", and this is the reason why you need to add **class="drag"** to any object that may be dragged and dropped (e.g. <img id=obj **class="drag"** src=cat.gif>).

```
if (event.srcElement.className=="drag")
```

The following line assigns the real ID of the object being dragged to **z**.

```
z=event.srcElement
```

Two variables **temp1** and **temp2** are added to represent the current (z.style.left, z.style.top) coordinates of the dragged object, as shown below:

```
temp1=z.style.pixelLeft
temp2=z.style.pixelTop
```

Notice that **pixelLeft** and **pixelTop** are new CSS positioning properties that specify or receive the **left** and **top** position in pixels respectively. A detailed discussion is available at a later lecture note.

Once a dragged object is checked and verified to have the **class="drag"** class name, the following statement triggers the **move()** function.

```
document.onmousemove=move
```

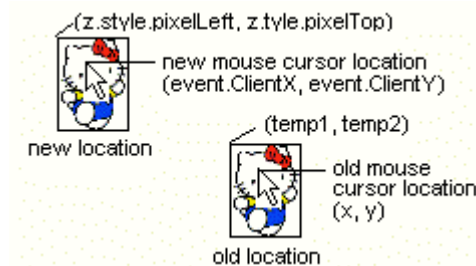
The **move()** function uses the following statement to detect if the left button of a mouse is still pressed and held.

```
if (event.button==1 && dragapproved){
.....
}
```

If the left button is pressed and held, the following statements update the values of (left, top) of the dragged object.

```
z.style.pixelLeft=temp1+event.clientX-x
z.style.pixelTop=temp2+event.clientY-y
```

The logic is the new values of (**z.style.pixelLeft**, **z.style.pixelTop**) equals to the old one (**temp1**, **temp2**) subtract the difference of new mouse cursor (**event.clientX**, **event.clientY**) and old mouse cursor (**x**, **y**).



## Speed

Moving object sometimes involves in the control of speed. In the Flying Witch game, for example, the bird moves faster than the witch does. In other words, these two objects have different speeds, as shown below.

```
function witch() {
.....
.....
setTimeout("witch()", 100);
}

function bird() {
.....
.....
setTimeout("bird()", 50);
}
```

The functions **setTimeout** and **setInterval** are similar, and they both can control the speed of a function. However, in fact, these two functions perform very differently. The **setTimeout** function delays for a specified time period and then triggers execution of a specified function.

Once the function is triggered the `setTimeout` has finished. You can terminate the execution of the `setTimeout` before it triggers the function by using the **clearTimeout** function.

In this game programming class, you frequently see `setTimeout` used at the end of a function to trigger another execution of the same function perhaps passing it different parameters. Where this is done the time from the first triggering of the function until the next will be the specified delay time plus the time taken to execute the function. Here is an example:

```
moreSnow();
function moreSnow() {
  // content of moreSnow function
  setTimeout("moreSnow()", speed);
}
```

The **setInterval** function also delays for a specified time before triggering the execution of a specific function. However, after triggering that function the command doesn't complete. Instead it waits for the specified time again and then triggers the function again and continues to repeat this process of triggering the function at the specified intervals until either the web page is unloaded or the `clearInterval` function is called.

The above code using **setTimeout** could have been written using **setInterval** instead and would have looped slightly faster since the loop would not have waited for the content of the function to be processed before triggering the repetition. Here is the above code rewritten to use `setInterval`:

```
moreSnow();
setInterval("moreSnow()", speed);
function moreSnow() {
  // content of moreSnow function
}
```

On the other hand, if you replace the `setTimeout` function with `setInterval`. When pressing any key. The code will fail. It is because pressing key will continuously trigger the `shoot()` function, but the `setInterval` doesn't complete a loop--it simply delays the looping. Starting a new loop before completing the old loop will fail the game.

```
function shoot() {

var i = b01.style.pixelLeft;

if (i <= 400) {
  b01.style.pixelLeft=i + 10;
  setTimeout("shoot()",20);
}
else {b01.style.display='none';
  b01.style.pixelLeft=95;
};
}
```

Which of the two functions you should choose to use depends on what you are trying to achieve. If different parameters need to be passed to the function on each call or the content of the function decides whether or not to call itself again then `setTimeout` is the one to use. Where the function just needs triggering at regular intervals then the `setInterval` function may produce simpler code within the function particularly if it has multiple exit points.

#### Directions

Objects may move in any direction, linear or non-linear. The term “linear” implies sequence, while “non-linear” implies skipping the sequence.

In the snake game (you created previously), the snake line moves in a **leaner** direction. Each time when you use arrow keys to change a direction, the line move point by point with an increment of

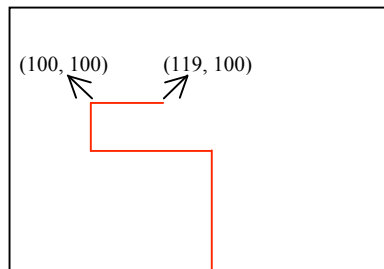
1, as illustrated below.

```

.....
case 37: n=1; k=0; break;
case 38: n=0; k=1; break;
case 39: n=-1; k=0; break;
case 40: n=0; k=-1; break;
.....
y-=k; x-=n;
codes = "<span style='position:absolute;left:"+x;
codes += "; top:"+ y +"'>.</span>";
area1.innerHTML += codes;
setTimeout("cc()", 0.1);
.....

```

If  $(x_1, y_1) = (100, 100)$  and you press the left arrow key (the key code is 37), then the very next dot is placed at  $(x_2, y_2) = (101, 100)$ , and the next one is  $(102, 100)$ ,  $(103, 100)$ ,..... In other words, to move from  $(100, 100)$  to  $(119, 100)$ . You need to move in a sequence of 100, 101, 102, 103, 104, 105, 106, ..., 117, 118, 119 along the  $x$  axis.



To handle a linear direction movement, simply use **increment** or **decrement**. In the ball bouncing example, two variables *i* and *j* will assign values of  $(x, y)$  to the **t1** object.

```

function dd() {

t1.style.left=i;
t1.style.top=j;
.....

i++; j+=Math.round(Math.random()*10);
.....
s1=setTimeout("dd()", 50);
}

```

where, *i++* means *i* is incremented by 1, which is equivalent to *i = i + 1*. In JavaScript, the increment and decrement operators act on only one operand. They are hence called Unary operators. They can be used either before or after a variable as in:

```

a++;
b--;

```

or

```

++a;
--b;

```

When these operators are used as prefixes, JavaScript first adds one to the variable and then returns the value. When used as suffixes, the value is first returned and then the variable is incremented.

When the increment or decrement is larger than 1, use the following syntax:

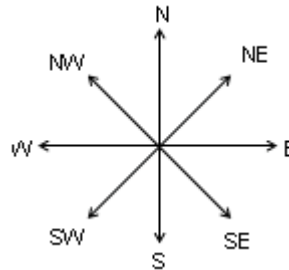
```
x += n; OR x -= n;
```

where  $n$  is a number, and  $n$  is not necessarily an integer. For example, in the Roulette Wheel game, there is a code block that uses increment that is  $\pi/180$  which is a decimal value since  $\pi$  is 3.1412.

```
if (a >= Math.PI *2) { a=0; }
else { a += Math.PI/180; }
```

You may need to move objects in directions such as northeast, southeast, northwest, and southwest. Assuming the object ID is **obj**, then the direction control in DHTMLs are (where  $n$  is a positive number):

- **NE:** `obj.style.pixelLeft + n; obj.style.pixelTop - n;`
- **SE:** `obj.style.pixelLeft + n; obj.style.pixelTop + n;`
- **NW:** `obj.style.pixelLeft - n; obj.style.pixelTop - n;`
- **SW:** `obj.style.pixelLeft - n; obj.style.pixelTop + n;`



When you want an object to jump from one location to another, and ignore all the points between the new and old locations, you are handling a movement of non-linear direction. For example, in the Flying UFO game, the **ufo** object's ( $x, y$ ) values are generated randomly by the **Math.random()** method each time when the **ufo\_fly()** function executes.

```
function ufo_fly() {
    ufo.style.pixelLeft = Math.round(Math.random()*screen.width);
    ufo.style.pixelTop = Math.round(Math.random()*screen.height);
    setTimeout("ufo_fly()", 500);
}
```

Consequently, the **ufo** object moves in a random unpredictable mode.

Moving along a circle

To move an object along a circle, as the ball in the Roulette Wheel game, you need to understand how computers create a circle.

According to geometry, a **circle** is the set of all points in a plane at a fixed distance (known as **radius**) from a fixed point (known as **center**).

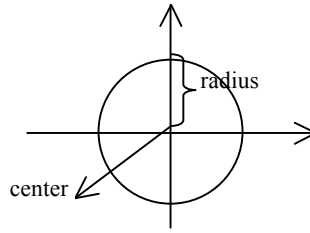
All the points on the circumference of the circle has a set of ( $x, y$ ) coordinates. The  $x$  value is determined by:

$$\text{centerX} + \text{radius} \times \sin\theta$$

and  $y$  is determined by:

$$\text{centerY} + \text{radius} \times \cos\theta$$

where  $\theta$  is incremented by  $\pi/180$ , and  $0 \leq \theta \leq 2\pi$ , because the circumference of a circle is  $2\pi$ .



Consider the following code, it uses (197, 197) as the (x, y) of the center and 120 as radius.

```
var r=120; a=0;
function cc() {
  b1.style.pixelLeft=197+r*Math.sin(a);
  b1.style.pixelTop=197+r*Math.cos(a);

  if (a >= Math.PI *2) { a=0; }
  else { a += Math.PI/180; }
  rotating=setTimeout("cc()", 2);
}
```

The line **b1.style.pixelLeft=197+r\*Math.sin(a);** is actually the expression of

$$\text{centerX} + \text{radius} \times \sin\theta$$

and **if (a >= Math.PI \*2) { a=0; }** is actually the DHTML expression of

$$0 \leq \theta \leq 2\pi$$

while **a += Math.PI/180;** means to increment by  $\pi/180$ . Consequently, each time when the **cc()** function executes, the object moves along the circumference of a circle that has the radius 120 and is centered at (197, 197).

#### Review Questions

1. In terms of DHTML games, \_\_\_ is responsible for the appearance of objects, while \_\_\_ is responsible for how objects move, behave, and interact with other objects.

- A. CSS, HTML
- B. HTML, CSS
- C. CSS, scripting language
- D. scripting, CSS

2. Given the following code block, which statement is correct?

```
function dd() {
  t1.style.left=i;
  t1.style.top=j;
  .....
}
```

- A. t1.style.top represents the x-coordinate of the t1 object.
- B. t1.style.left represents the y-coordinate of the t1 object.
- C. t1.style.left represents the x-coordinate of the t1 object.
- D. t1.style.top is equivalent to t1.getElementById().style.top

3. Which is equivalent to the following line?

```
var x=m1.style.pixelLeft;
```

- A. `var x=document.getElementById('m1').style.pixelLeft;`
- B. `var x=m1.getElementById().style.pixelLeft;`
- C. `var x=m1.getElementById('m1').style.pixelLeft;`
- D. `var x=document.getElementById().m1.style.pixelLeft;`

4. Given the following code, which statement is correct?

```
function change_color() {
var oldID = event.srcElement.id;
document.getElementById(oldID).style.color=oldID;
}
```

- A. `oldID` will represent the ID of the object that triggers the event.
- B. `oldID` represents the previous ID of the object that triggers the event, because `srcElement.id` will assign a new ID to it.
- C. `event.srcElement.id` retrieve the ID of the mouse button.
- D. `event.srcElement.id` retrieve the key code of the mouse button.

5. Which statement is correct?

- A. The default position is "absolute".
- B. The `srcElement` property sets or retrieves the object that fired the event.
- C. The `display` property sets if an element should be visible or invisible.
- D. The `visibility` property sets how/if an element is displayed

6. Which sets an object's position according to the normal flow, and then offset by the `op` and `left` properties?

- A. `position:static`
- B. `position:absolute`
- C. `position:relative`
- D. `position:physical`

7. Which item will be on the front?

- A. `<img id="m1" style=z-index: 1>`
- B. `<img id="m2" style=z-index: 2>`
- C. `<img id="m3" style=z-index: 3>`
- D. `<img id="m4" style=z-index: 4>`

8. Which can control the speed of a function in JavaScript?

- A. `speed();`
- B. `setTimeout();`
- C. `controlTime();`
- D. `setSpeed();`

9. Given an object with ID "t1", which can move this object toward northwest?

- A. `t1.style.pixelLeft + 2; t1.style.pixelTop + 2;`
- B. `t1.style.pixelLeft + 2; t1.style.pixelTop - 2;`
- C. `t1.style.pixelLeft - 2; t1.style.pixelTop - 2;`
- D. `t1.style.pixelLeft - 2; t1.style.pixelTop + 2;`

10. Given a center coordinate (100, 150) and a radius 25. Which represents the y value of any point on the circle?

- A.  $100 + 25 * \sin \theta$
- B.  $150 + 25 * \sin \theta$
- C.  $100 + 25 * \cos \theta$

D.  $150 + 25 * \cos \theta$

## Game Programming

### Lab #7

#### Preparation #1:

1. Create a new directory named **C:\games**.
2. Use Internet Explorer to go to **<http://business.cypresscollege.edu/~pwu/cis261/download.htm>** to download lab7.zip (a zipped) file. Extract the files to C:\games directory.

#### Learning Activity #1: Drag and Drop

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named **C:\games\lab7\_1.htm** with the following contents:

```
<html>

<style>
.drag {position:absolute}
</style>
<script src="dragndrop.js"></script>

<script>
var codes = "";
var x, y;

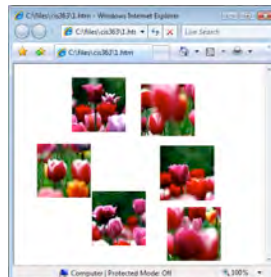
function init() {
  for (i=1; i<=6; i++) {

    x = Math.floor(Math.random()*document.body.clientWidth/2);
    y = Math.floor(Math.random()*document.body.clientHeight/2);

    codes += "<img id='cy" + i + "' class='drag'";
    codes += " src=cy" + i + ".jpg style='left: "+x+";top:"+y+"'><br>";
  }

  areal.innerHTML = codes;
}
</script>
<body onLoad=init()>
<span id=areal></span>
</body>
</html>
```

3. Test the program. A sample output looks:



## Learning Activity #2: USA Map

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab7\_2.htm with the following contents:

```
<html>

<style>
.drag {position:absolute}
</style>

<script src="dragndrop.js"></script>

<body>

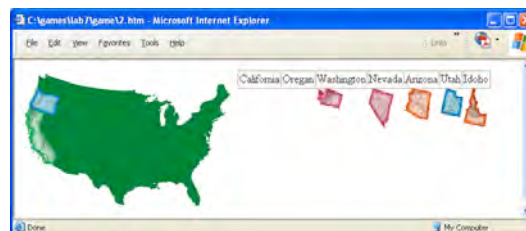
<table class="drag" border=1 cellspacing=0>
<tr>
<td>California</td>
<td>Oregon</td>
<td>Washington</td>
<td>Nevada</td>
<td>Arizona</td>
<td>Utah</td>
<td>Idaho</td>
</tr>

<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>

</table>

</body>
</html>
```

3. Test the program. A sample output looks:



## Learning Activity #3: Roulette Wheel 2

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab7\_3.htm with the following contents:

```
<html>

<script>
var i=1;
function spin() {
  if (i>=16) {i=1; }
  else {
    i++;
  }
  m1.src="wheel_" + i + ".gif";
  spinning = setTimeout("spin()", 100);
}

var r=120; a=0;
function cc() {
  b1.style.left=197+r*Math.sin(a);
  b1.style.top=197+r*Math.cos(a);

  if (a >= Math.PI *2) { a=0; }
  else { a += Math.PI/180; }
  rotating=setTimeout("cc()", 2);
}
</script>

<body>




<div style="position:absolute; left:50; top:400;">
<button onClick="cc();spin()">Rotate</button>
<button onClick="clearTimeout(rotating); clearTimeout(spinning)">Stop</button>
</div>

</body>
</html>
```

3. Test the program. A sample output looks:



#### Learning Activity #4: Running Bugs

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named **C:\games\lab7\_4.htm** with the following contents:

```
<html>

<style>
.ants {position:absolute}
</style>

<script>
function init() {
  for (i=1; i<=8; i++) {
    x=Math.floor(Math.random()*document.body.clientWidth);
    y=Math.floor(Math.random()*document.body.clientHeight);

    codes = "<img src=ant"+i+".gif id=a"+i+" class=ants"
    codes += " style='left:"+x+";top:"+y+"'>";
    area1.innerHTML += codes;
  }
}

function move() {
a1.style.pixelLeft -= 5;
a1.style.pixelTop += 5;

a2.style.pixelTop += 5;

a3.style.pixelLeft += 5;
a3.style.pixelTop += 5;

a4.style.pixelLeft -= 5;

a5.style.pixelLeft += 5;

a6.style.pixelLeft -= 5;
a6.style.pixelTop -= 5;

a7.style.pixelTop -= 5;

a8.style.pixelLeft += 5;
a8.style.pixelTop -= 5;

if (a1.style.pixelLeft <= 0) {a1.style.pixelLeft = document.body.clientWidth; }
if (a1.style.pixelTop >= document.body.clientHeight) {a1.style.pixelTop = 0; }

if (a2.style.pixelTop >= document.body.clientHeight) {a2.style.pixelTop = 0; }

if (a3.style.pixelLeft >= document.body.clientWidth) {a3.style.pixelLeft = 0; }
if (a3.style.pixelTop >= document.body.clientHeight) {a3.style.pixelTop = 0; }

if (a4.style.pixelLeft <= 0) {a4.style.pixelLeft = document.body.clientWidth; }

if (a5.style.pixelLeft >= document.body.clientWidth) {a5.style.pixelLeft = 0; }

if (a6.style.pixelLeft <= 0) {a6.style.pixelLeft = document.body.clientWidth; }
if (a6.style.pixelTop <= 0) {a6.style.pixelTop = document.body.clientHeight; }

if (a7.style.pixelLeft <= 0) {a7.style.pixelLeft = document.body.clientWidth; }

if (a8.style.pixelLeft >= document.body.clientWidth) {a8.style.pixelLeft = 0; }
if (a8.style.pixelTop <= 0) {a8.style.pixelTop = document.body.clientHeight; }
```

```

setTimeout("move()", 20);
}
</script>

<body onLoad="init();move()">

<script id=area2></script>
<span id=areal></span>

</body>

</html>

```

3. Test the program. A sample output looks:



### Learning Activity #5: Rush Hour Game

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab7\_5.htm with the following contents:

```

<html>
<style>
.dragr {position:absolute}
.dragc {position:absolute}
</style>

<script>

var dragapproved=false
var z,x,y
function move(){
  if (event.button==1 && dragapproved) {

var clsName = event.srcElement.className;
switch (clsName) {
  case "dragr":
    z.style.pixelLeft=temp1+event.clientX-x;
    break;
  case "dragc":
    z.style.pixelTop=temp2+event.clientY-y
    break;
}
return false
}
}

function drags(){
var clsName = event.srcElement.className;

```

```

if (!document.all) { return }

if ((clsName=="dragr") || (clsName=="dragc")) {
    dragapproved=true
    z=event.srcElement
    temp1=z.style.pixelLeft
    temp2=z.style.pixelTop
    x=event.clientX
    y=event.clientY
    document.onmousemove=move
}
}

<!-- generating table -->
function cc() {
codes = "<table cellpadding=2 border=1";
codes += " style='left:0; top:0; position:absolute'>";
for (h=1; h<=8; h++) {
    codes += "<tr height=36>";

    for (i=1; i<=8; i++) {
        codes += "<td width=36>&nbsp;</td>";
    }

    codes += "</tr>";
}
codes += "</table>";
areal.innerHTML = codes;
}

document.onmousemove=move
document.onmousedown=drags
document.onmouseup=new Function("dragapproved=false")

</script>
<body onLoad=cc()>
<div id=areal></div>



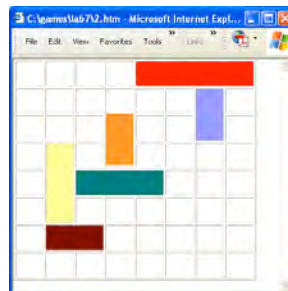




</body>
</html>

```

3. Test the program. A sample output looks:



### Submittal

Upon completing all the learning activities,

1. Upload all files you created in this lab to your remote web server.
2. Log in to Blackboard, launch Assignment 07, and then scroll down to question 11.
3. Copy and paste the URLs to the textbox. For example,
  - [http://www.geocities.com/cis261/lab7\\_1.htm](http://www.geocities.com/cis261/lab7_1.htm)
  - [http://www.geocities.com/cis261/lab7\\_2.htm](http://www.geocities.com/cis261/lab7_2.htm)
  - [http://www.geocities.com/cis261/lab7\\_3.htm](http://www.geocities.com/cis261/lab7_3.htm)
  - [http://www.geocities.com/cis261/lab7\\_4.htm](http://www.geocities.com/cis261/lab7_4.htm)
  - [http://www.geocities.com/cis261/lab7\\_5.htm](http://www.geocities.com/cis261/lab7_5.htm)

No credit is given to broken link(s).

## Game Programming

### Lecture #8 Collision detection and response

**Introduction** Collision detection and collision response is the act of detecting whether or not a collision has happened, and if so, responding to the collision. In physical simulations, video games and computational geometry, collision detection includes algorithms from checking for intersection between two given solids, to calculating trajectories, impact times and impact points in a physical simulation. There are several methods of collision detection, and the right one for the job depends on the shape of the object.

DHTML allows developers to create client-side applications of a type that was once possible only with server-side programming. By combining JavaScript with Cascading Style Sheets, it's possible to perform effective collision detection.

The following JavaScript example uses two screen elements (which could be images, layers, divisions, or anything else in the document object hierarchy) called **myObject1** and **myObject2**.

The variables *x* and *y* hold either a true or a false value, depending on whether the two objects overlap on either the horizontal plane or the vertical plane. If both *x* and *y* evaluate to true, then the two objects have collided.

```
x = ((myObject1.style.posLeft >= myObject2.style.posLeft) &&
(myObject1.style.posLeft <= myObject2.style.posLeft));

y = ((myObject1.style.posTop >= myObject2.style.posTop) &&
(myObject1.style.posTop <= myObject2.style.posTop));

if (x && y)
{
    // Collision has occurred.
}
```

The **move\_crab()** function, for example, use the *if* statement with programming logic similar to the above one.

```
function move_crabs() {
var redSpeed = Math.floor(Math.random()*20);
.....

red.style.pixelLeft += redSpeed;
.....

if (red.style.pixelLeft + 32 >= area2.style.pixelLeft) {
    clearTimeout(s1);alert("Winner is the red crab!");}
.....
}
```

The following line is the spine of the collision detection, it determine whether or not the **red** object and the **area2** object collide.

**One-dimensional collision detection** Handling collision detection in 2D graphics is based on a simple idea--determine whether or not two or more graphics collide with one another. In a Web game, for example, graphics are in a shape of rectangle, so you can simply try to detect whether two or more rectangular areas are in any way touching or overlapping each other.

A rectangular image has four vertexes. In the following graphic, 1, 2, 3, and 4 represent each of

the vertexes:



Assuming this graphic is given an object ID **obj1**, the (x, y) coordinates of these 4 vertexes can be represented by:

- 1: (obj1.style.pixelLeft, obj1.style.pixelTop)
- 2: (obj1.style.pixelLeft + obj1.style.width, obj1.style.pixelTop)
- 3: (obj1.style.pixelLeft + obj1.style.width, obj1.style.pixelTop + obj1.style.height)
- 4: (obj1.style.pixelLeft, obj1.style.pixelTop + obj1.style.height)

A simple implementation of this method is as follows:

```
<html><style>
.drag {position: absolute}
</style>

<script>
function obj1Move() {
obj1.style.left = obj1.style.pixelLeft + 5;
s1 = setTimeout("obj1Move()", 50);
}

function obj2Move() {
obj2.style.left = obj2.style.pixelLeft + 5;

if (obj2.style.pixelLeft >= obj1.style.pixelLeft) {
clearTimeout(s1);
clearTimeout(s2);
p1.innerText = "Game Over!";
}

else {
s2 = setTimeout("obj2Move()", 20);
}
}
</script>

<body onLoad="obj1Move();obj2Move();">

<img id="obj2" class="drag" src='gst1.gif' style="top:10;left:10">
<p id=p1 class="drag" style="top:150; left:140;"></p>

</body></html>
```

There are two image files: **pac1.gif** and **gst1.gif**. Their IDs are **obj1** and **obj2** respectively. In the obj1Move() function, the **obj1** is set to increment its value of x-coordinate by 5px every 50 milliseconds, as shown below:

```
obj1.style.left = obj1.style.pixelLeft + 5;
```

Similarly, the **obj2** is set to increment its value of x-coordinate by 5px every 20 milliseconds, as shown below:

```
obj2.style.left = obj2.style.pixelLeft + 5;
```

Since obj2 moves faster than obj1, obj2 will soon catch up and overlaps with obj1. You can use the following code block to detect such collision, and the logic is “if the x-coordinate of obj2 is greater than or equal to the x-coordinate of obj1”:

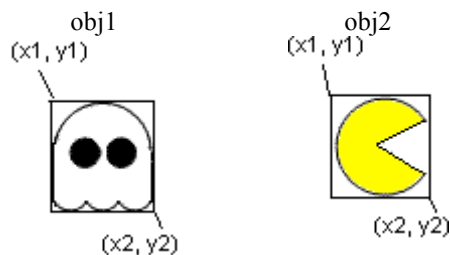
```
if (obj2.style.pixelLeft >= obj1.style.pixelLeft)
```

Two-dimensional collision detection

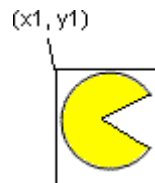
In a given plane, rectangles are two-dimensional: vertical and horizontal. In many games, collisions can happen in any direction, vertically, horizontally, or both. The following figure illustrates four of the possible directions these two image files overlap with each other.



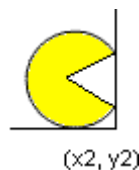
A rectangular area has four vertices, but you can use the diagonal to define the perimeter of the rectangle. This diagonal is drawn by connecting the top-left-most point and bottom-right-most point. The following figure uses (x1, y1) and (x2, y2) to indicate the four coordinate sets of each image (obj1 and obj2).



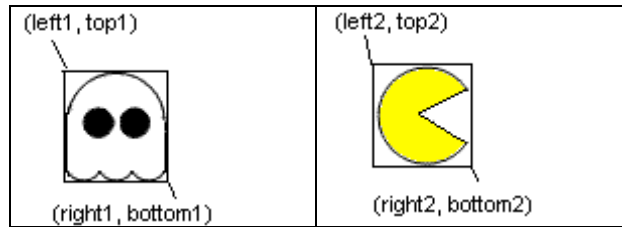
When the value of x1 increases, the object moves from left to right, and vice versa. When the value of y1 increases, the object moves from top to bottom, and vice versa. Therefore, the set of (x1, y1) take care of the left and top border.



Similarly, the change in values of (x2, y2) handles the right and bottom border.



Once you understand the logic, you can write codes to detect the collision of these two images from any direction. First, you can try to define four variables to represent (x1, y1) and (x2, y2) of both image file:



The values of (x1, y1) of obj1 are assigned to (left1, top1) by the following statements:

```
left1 = obj1.style.pixelLeft;
top1 = obj1.style.pixelTop;
```

The value of (x2, y2) can be easily determined, they are:

- $x2 = x1 + \text{width}$
- $y2 = y1 + \text{height}$

So you can translate them into:

```
right1 = obj1.style.pixelLeft + obj1.style.width;
bottom1 = obj1.style.pixelTop + obj1.style.height;
```

The following is an example of how you handle the coding in DHTML.

```
<script>

var left1, left2;
var right1, right2;
var top1, top2;
var bottom1, bottom2;

left1 = obj1.style.pixelLeft;
left2 = obj2.style.pixelLeft;

right1 = obj1.style.pixelLeft + obj1.style.width;
right2 = obj2.style.pixelLeft + obj2.style.width;

top1 = obj1.style.pixelTop;
top2 = obj2.style.pixelTop;

bottom1 = obj1.style.pixelTop + obj1.style.height;
bottom2 = obj2.style.pixelTop + obj2.style.height;

.....
.....

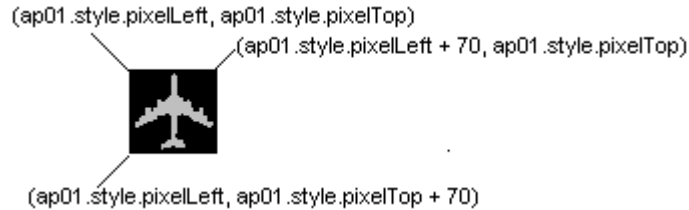
function obj2Move() {
obj2.style.left = obj2.style.pixelLeft + 5;

if (((bottom1 < top2) || (top1 > bottom2) || (right1 < left2)
|| (left1 > right2)) {
clearTimeout(s1);
clearTimeout(s2);
p1.innerText = "Game Over!";
}

else {
s2 = setTimeout("obj2Move()", 20);
}
}
```

```
</script>
```

In the Jet Fighter game, the **check()** function handles the collision detection. The **ap01** object is the enemy's airplane, while **b01** is the bullet.



The **check()** function uses a simple logic. First, the **b01**'s **pixelLeft** must be between **ap01.style.pixelLeft** and **ap01.style.pixelLeft + 70**. Second, the **b01**'s **pixelTop** must be less than or equal to **ap01.style.pixelTop**. Notice that the dimension of **ap01** is  $70 \times 70$ , meaning both the width and height are 70px.

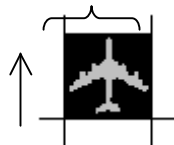
```
function check() {
  if ((b01.style.pixelLeft >= ap01.style.pixelLeft) &&
      (b01.style.pixelLeft <= ap01.style.pixelLeft + 70) &&
      (b01.style.pixelTop <= ap01.style.pixelTop + 70))
  {
    ap01.src="explode.gif";
    setTimeout("ap01.style.display='none'",1000);
  }
}
```

The following lines check if the **b01**'s **pixelLeft** is between **ap01.style.pixelLeft** and **ap01.style.pixelLeft + 70**:

```
(b01.style.pixelLeft >= ap01.style.pixelLeft) &&
(b01.style.pixelLeft <= ap01.style.pixelLeft + 70)
```

The following line check if the less than or equal to **ap01.style.pixelTop**:

```
(b01.style.pixelTop <= ap01.style.pixelTop + 70)
```



When all the three conditions are true, the following line changes the image of **ap01** to another one that is an animated flame.

```
ap01.src="explode.gif";
```



**Over-detecting** The method usually works fairly well to rectangles. But it has a problem of “over-detecting” collisions. For example, the Pacman graphic is not rectangular, it is roundish. When it is put in a rectangular image file, as shown below, there will be a distance between each vertex and the graphics. The blue area in the following figure represents the blank space.



Because the roundish shape of the graphics, the vertexes of each image file contains empty space. But the bounding rectangles of each image are clearly intersecting, so a traditional collision detecting algorithm as the above code uses will have deviations.

One acceptable solution to this over-detecting problem is to define a somewhat smaller rectangle than the full extents of the image, and use this smaller rectangle for the collision detection.

A good idea, in this case, is to use 80% of the full bounding box of the image. To do so, you need to remove 20% of the space from the rectangular image. You can define six new variables: **obj1Width**, **obj1Height**, **obj1ColWidth**, **obj1ColHeight**, **obj\_X\_offset**, and **obj\_Y\_offset**.

```
obj1Width = obj1.style.pixelWidth;
obj1Height = obj1.style.pixelHeight;

obj1ColWidth = objWidth * 0.8;
obj1ColHeight = objHeight * 0.8;

obj1_X_offset = (objWidth - objColWidth) / 2;
obj1_Y_offset = (objHeight - objColHeight) / 2;
```

During initialization, wherever the above code retrieves the width and height of **obj1**, and uses them to calculate the **obj1ColWidths** and **obj1ColHeights** to be 20% smaller. It also defines the offset fields to describe where to set the bounding box relative to the object's coordinates.

#### Advanced collision detection

In a previous lecture, you learned the techniques of handling drag and drop. Similar technique can be used to detect collisions. For example, when the monsters are bumping into PacMan, you can check whether the floating DIV element is touching or overlapping a valid Drag Target.

assuming,

```
<DIV class="drag">
  <IMG SRC="mind.gif" WIDTH=115 HEIGHT=45 BORDER=0>
</DIV>
```

As in the previous technique, there is nothing unusual about the DIV element, except that it has been given a class name. Of course, you would set the WIDTH and HEIGHT attributes of the IMG to fit your images. It is important that all IMGs have the same WIDTH and HEIGHT. Repeat this declaration for each DIV you want to make draggable. Finally add an absolutely positioned DIV to your page like this:

```
<DIV ID="floater" style="visibility:hidden;position:absolute">
  <IMG SRC="trans.gif" WIDTH=115 HEIGHT=45 BORDER=0>
</DIV>
```

This floating DIV is the only moving element in the page, the rest of the DIVs are not absolutely positioned, and therefore cannot be moved. Also notice that the IMG inside the DIV is given a SRC, even though it is initially invisible. This is because of a bug in IE4 that prevents dynamically setting the SRC of an IMG if the original SRC is not specified.

The reason we need this algorithm is that since the mouse is over the floating DIV, the onmouseup event's srcElement will not be accurate. Also, since the surface area of the floating DIV is quite large, it is quite possible that the DIV may overlap two or more potential targets.

Only by comparing the area of overlap of each target, can we accurately determine which the target the user intended was. Happily, the Collision Detection Algorithm can do this.

The algorithm goes something like this:

```
Given 2 rectangular objects of the same size,
the Horizontal and Vertical overlap between
them is given by :
hOverlap = W - |x2-x1|
vOverlap = H - |y2-y1|
```

```
where
W=width of the objects
H=Height of the objects
x1,y1=Top Left co-ordinates of Obj 1
x2,y2=Top Left co-ordinates of Obj 2
```

```
if both hOverlap and vOverlap are positive
their product gives the area of overlap of
the 2 objects.
if either or both are negative, the objects
do not overlap (have not collided).
```

Note: This algorithm was devised by Stanley Rajasingh.

Now that we have the algorithm, coding the **onMouseup** handler is a trivial exercise. Looking at it we see -

```
function DragEnd(){
    if(!DragEl) return;
    X1=floater.style.pixelLeft;
    Y1=floater.style.pixelTop;
    MaxArea=0;
    TargetElem=null;

    Targets=document.all;
    for(i=0;i<Targets.length;i++){
        if(Targets[i].className=="dnd"){
            X2=getRealPos(Targets[i].children[0],"Left");
            Y2=getRealPos(Targets[i].children[0],"Top");
            hOverlap=floater.offsetWidth-Math.abs(X2-X1);
            vOverlap=floater.offsetHeight-Math.abs(Y2-Y1);

            areaOverlap=(hOverlap>0)&&(vOverlap>0)?hOverlap*vOverlap:0;
            if(areaOverlap>MaxArea){
                MaxArea=areaOverlap;
                TargetElem=Targets[i];
            }
        }
    }

    floater.style.visibility='hidden';

    if(TargetElem){
        //Swap Images
        DragEl.src=TargetElem.children[0].src;
        TargetElem.children[0].src=floater.children[0].src;
    } else {
        //Restore Original Image
        DragEl.src=floater.children[0].src;
        return;
    }
    DragEl=null;
}
```

```
}
```

The magic *for* loop cycles through each element in the above code is that it checks to see if it is a valid `Drag Target (className=="dnd")`. If it is, the code applies the Collision Detection Algorithm to obtain the overlap area. The element which has the largest overlap with the floating DIV becomes the **TargetElem**.

Once the target has been obtained, the function swaps the source and target images, and then hides the floating DIV, creating the illusion that the **Drag Source** and **Drag Target** have exchanged places. Finally it resets the Drag Source pointer (**DragEl**) to null, indicating that the dragging has ended.

Be aware that since this algorithm takes over the **onMouseOver**, **onMouseUp**, and **onMouseMove** handlers of the document, any other element on the page that uses these events, such as a DHTML menu, will not function. In order to prevent this you might want to overload these handlers rather than taking over them entirely.

**Collision Math** Given two objects **X** and **Y** with position vectors  $r_x$  and  $r_y$ , you want to be able to test whether or not the objects are collided as a function of  $r_x - r_y$ .

The solution is that objects **X** and **Y** overlap if the set

$$\{ (x, y) \mid x \in X \wedge y \in Y \wedge x + r_x = y + r_y \}$$

is non-empty. This is equivalent to the condition of

$$r_x - r_y \in Z$$

where **Z** is the set

$$\{ y - x \mid x \in X \wedge y \in Y \}$$

If **X** and **Y** are arbitrary shapes, then the shape **Z** should be pre-computed. However there is no easy way to efficiently determine membership of a point in an arbitrary shape in JavaScript (unless perhaps you separately encode the shape **Z** as a suitable array of arrays).

One can avoid this problem by making all sprites a rectangular shape, in which case if **X** is a rectangle with width  $w_x$  and height  $h_x$  and **Y** is a rectangle with width  $w_y$  and height  $h_y$ , then **Z** will be rectangle with width  $w_x + w_y$  and height  $h_x + h_y$ . Which explains the following code:

```
function collided (element1, element2) {
  var x1 = element1.offsetLeft;
  var x2 = element2.offsetLeft - element1.offsetWidth;
  var x3 = element2.offsetLeft + element2.offsetWidth;
  if (x2 <= x1 && x1 <= x3) {
    var y1 = element1.offsetTop;
    var y2 = element2.offsetTop - element1.offsetHeight;
    var y3 = element2.offsetTop + element2.offsetHeight;
    return y2 <= y1 && y1 <= y3;
  }
  else {
    return false;
  }
}
```

In the above code, **element1** is **X**, **element2** is **Y**, and the variables **x2**, **x3**, **y2** and **y3** represent the boundaries of **element2** as if it had a size of the hypothetical **Z** object.

The other thing to watch with collision detection is that if you are checking it once per game loop,

then you want to make sure that the objects aren't moving towards each other so fast that they can skip past each other, i.e. their total distance travelled relative to each other per game loop should not exceed the size of the vector corresponding to their combined width and height.

## Review Questions

1. Handling collision detection in 2D graphics is based on determining \_\_\_\_\_.
  - A. if two objects' shapes are symmetrical.
  - B. whether or not two or more objects collide with one another.
  - C. if two objects' motions are synchronous.
  - D. whether or not two or more objects resemble one another.

2. Which can express the following sentence:

"If the x-coordinate of obj2 is greater than or equal to the x-coordinate of obj1"

- A. if ((obj2.style.pixelLeft > z.style.pixelLeft) && (obj2.style.pixelLeft = z.style.pixelLeft))
- B. if (z.style.pixelLeft >= obj2.style.pixelLeft)
- C. if (obj2.style.pixelLeft >= z.style.pixelLeft)
- D. if ((z.style.pixelLeft > obj2.style.pixelLeft) && (z.style.pixelLeft = obj2.style.pixelLeft))

3. Given the following rectangular object with an ID "z", which can represent the (x, y) coordinate of the vertex 3 using DHTML?

```

1---2
|   |
|   |
4---3

```

- A. (z.style.pixelLeft + object.style.width, z.style.pixelTop)
- B. (z.style.pixelLeft, z.style.pixelTop)
- C. (z.style.pixelLeft, z.style.pixelTop + object.style.height)
- D. (z.style.pixelLeft + object.style.width, z.style.pixelTop + object.style.height)

4. Given the following rectangular object with an ID "z", which can represent the (x, y) coordinate of the vertex 2 using DHTML?

```

1---2
|   |
|   |
4---3

```

- A. (z.style.pixelLeft + object.style.width, z.style.pixelTop)
- B. (z.style.pixelLeft, z.style.pixelTop)
- C. (z.style.pixelLeft, z.style.pixelTop + object.style.height)
- D. (z.style.pixelLeft + object.style.width, z.style.pixelTop + object.style.height)

5. Given an object that has dimension of 60 X 60 and the ID "b", which can represent the (x, y) coordinate of the vertex 4 using DHTML?

```

1---2
|   |
|   |
4---3

```

- A. (b.style.pixelLeft + 60, b.style.pixelTop + 60)
- B. (b.style.pixelLeft, b.style.pixelTop + 60)
- C. (b.style.pixelLeft + 60, b.style.pixelTop)
- D. (b.style.pixelLeft, b.style.pixelTop)

6. Which can determine if the less than or equal to `ap01.style.pixelTop`?
- A. `if (b01.style.pixelTop <= ap01.style.pixelTop)`
  - B. `if (ap01.style.pixelTop <= b01.style.pixelTop)`
  - C. `if ((ap01.style.pixelTop < b01.style.pixelTop) && (ap01.style.pixelTop < b01.style.pixelTop))`
  - D. `if ((b01.style.pixelTop < ap01.style.pixelTop) && (b01.style.pixelTop < ap01.style.pixelTop))`
7. Which can determine if the `b01`'s `pixelLeft` is between `ap01.style.pixelLeft` and `ap01.style.pixelLeft + 70`?
- A. `if (b01.style.pixelTop <= ap01.style.pixelTop + 70)`
  - B. `if ((b01.style.pixelLeft >= ap01.style.pixelLeft) && (b01.style.pixelLeft <= ap01.style.pixelLeft + 70))`
  - C. `if ((b01.style.pixelLeft <= ap01.style.pixelLeft) && (b01.style.pixelLeft >= ap01.style.pixelLeft + 70))`
  - D. `if ((b01.style.pixelLeft > ap01.style.pixelLeft) && (b01.style.pixelLeft < ap01.style.pixelLeft + 70) && (b01.style.pixelLeft = ap01.style.pixelLeft))`
8. Which can force the "z" object to change its value of the "src" property to "apple.gif"?
- A. `z.src = "apple.gif"`
  - B. `z.style.src = "apple.gif"`
  - C. `z.style.Src = "apple.gif"`
  - D. `z = "apple.gif"`
9. The term "over-detecting" means \_\_\_\_.
- A. a rectangular image file with over-sized graphics that cause unexpected collision.
  - B. a rectangular image file with over-programmed code to make the collision detection function over sensitive.
  - C. a rectangular image file that contains polygons that makes the collision detection function over sensitive.
  - D. a rectangular image file that contains graphics that does not fully fill the rectangular area, such that the unfilled area become blank space when collision happens.
10. What statement is correct about "collision detection and response"?
- A. It is the act of detecting whether or not a collision has happened.
  - B. It uses algorithms from checking for intersection between two given solids.
  - C. Collision detection is responsible for determining whether or not a collision happens, while collision response handles the counter-action when the collision is detected.
  - D. All of the above.

## Game Programming

### Lab #8

### Collision detection and response

#### Preparation #1:

1. Create a new directory named **C:\games**.
2. Use Internet Explorer to go to **<http://business.cypresscollege.edu/~pwu/cis261/download.htm>** to download lab8.zip (a zipped) file. Extract the files to C:\games directory.

#### Learning Activity #1: Simple Pacman I

1. Change to the C:\games directory.
2. Use Notepad to create a new file named **C:\games\lab8\_1.htm** with the following contents:

```
<html>
<style>
.drag {position: absolute}
</style>

<script>
function obj1Move() {
obj1.style.left = eval(obj1.style.left.replace('px','')) + 5;
s1 = setTimeout("obj1Move()", 50);
}

function obj2Move() {
obj2.style.left = eval(obj2.style.left.replace('px','')) + 5;

if (eval(obj2.style.left.replace('px','')) >=
    eval(obj1.style.left.replace('px',''))) {
    clearTimeout(s1);
    clearTimeout(s2);
    p1.innerText = "Game Over!";
}

else {
    s2 = setTimeout("obj2Move()", 20);
}
}

</script>

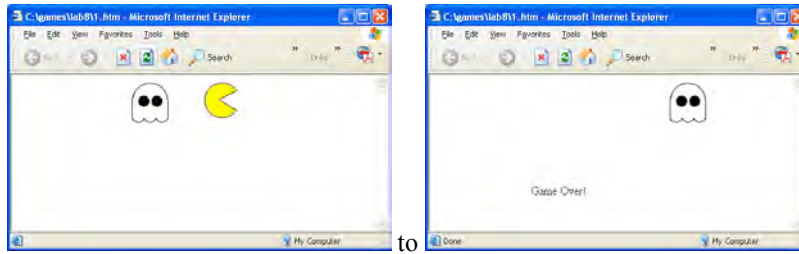
<body onLoad="obj1Move();obj2Move();">



<p id=p1 class="drag" style="top:150; left:140;"></p>

</body>
</html>
```

3. Test the program. Obj2 chases Obj1, and when they overlap each other, the game is over. Click Refresh to restart the game. A sample output looks:



## Learning Activity #2: Simple Pacman II

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named **C:\games\lab8\_2.htm** with the following contents:

```
<html>
<style>
.drag {position: absolute}
</style>

<script src="dragndrop.js"></script>
<script>

var left1, left2;
var top1, top2;

function obj1Move2Right() {
left1 = eval(obj1.style.left.replace('px',''));
top1 = eval(obj1.style.top.replace('px',''));

if (left1 >= (document.body.clientWidth - 100)) {
clearTimeout(ToRight);
obj1Move2Left();
}

else {
obj1.src="pac1.gif"
obj1.style.left = left1 + 2;
eat();
ToRight = setTimeout("obj1Move2Right()", 20);
}
}

function obj1Move2Left() {
left1 = eval(obj1.style.left.replace('px',''));
top1 = eval(obj1.style.top.replace('px',''));

if (left1 <= 0) {
clearTimeout(ToLeft);
obj1Move2Right();
}

else {
obj1.src="pac2.gif"
obj1.style.left = left1 - 2;
ToLeft = setTimeout("obj1Move2Left()", 20);
}
}
}
```

```

function dots() {
    scripts ="function eat() {";

    k = (document.body.clientWidth - 100) / 20;
    for (i=1; i<=k; i++) {
        codes = "<img id=d"+i+ " src='dot.gif' class='drag'";
        codes += " style='z-index:1;top:30; left:"+ i*20+"'>";
        areal.innerHTML += codes;
    }
}

</script>
<script src="eat.js"></script>

<body onLoad="dots();obj1Move2Right();">

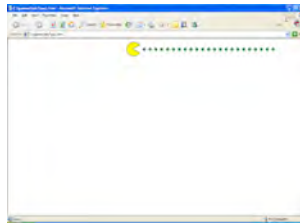


<span id=areal></span>

</body>
</html>

```

3. Test the program. A sample output looks:



### Learning Activity #3: Racing Crabs

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab8\_3.htm with the following contents:

```

<html>

<style>
div, img { position: absolute;left:10px }
</style>

<script>
function init() {
    areal.style.width = document.body.clientWidth - 80;
    areal.style.borderRight = "solid 2 white";

    area2.style.pixelLeft = document.body.clientWidth - 70;
}

function move_crabs() {
    var redSpeed = Math.floor(Math.random()*20);
    var greenSpeed = Math.floor(Math.random()*20);
    var blueSpeed = Math.floor(Math.random()*20);
    var whiteSpeed = Math.floor(Math.random()*20);

    red.style.pixelLeft += redSpeed;
    green.style.pixelLeft += greenSpeed;
    blue.style.pixelLeft += blueSpeed;
}

```

```

white.style.pixelLeft += whiteSpeed;

if (red.style.pixelLeft + 32 >= area2.style.pixelLeft) {
    clearTimeout(s1);alert("Winner is the red crab!");}
else if (green.style.pixelLeft + 32 >= area2.style.pixelLeft) {
    clearTimeout(s1);alert("Winner is the green crab!");}
else if (blue.style.pixelLeft + 32 >= area2.style.pixelLeft) {
    clearTimeout(s1);alert("Winner is the blue crab!");}
else if (white.style.pixelLeft + 32 >= area2.style.pixelLeft) {
    clearTimeout(s1);alert("Winner is the white crab!");}
else { s1 = setTimeout("move_crabs()", 200); }
}

</script>

<body onLoad=init()>

<div style="background-color:black; width:100%; height:130; z-index:2" id="area1">
<br>
<br>
<br>
<br>
</div>

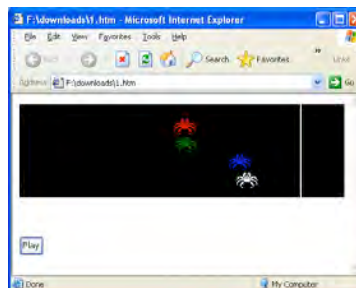
<div style="background-color:black; width:60; height:130; z-index:1"
id="area2"></div>

<button style="position:absolute; top:200;" onClick="move_crabs()">Play</button>

</body>
</html>

```

3. Test the program. Click the Play button to start. Click the browser's Refresh button to reset. A sample output looks:



#### Learning Activity #4: Jet Fighter

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab8\_4.htm with the following contents:

```

<html>
<SCRIPT>

function init() {
ap01.style.left=Math.floor(Math.random() * document.body.clientWidth)-120;

st01.style.left=Math.floor(Math.random() * document.body.clientWidth);
st01.style.top=Math.floor(Math.random() * document.body.clientHeight);

```

```

if ((st01.style.pixelTop <= 100) || (st01.style.pixelTop >=
document.body.clientHeight)) {
    st01.style.top=Math.floor(Math.random() * document.body.clientHeight); }

b01.style.pixelLeft=st01.style.pixelLeft + 35;
b01.style.pixelTop=st01.style.pixelTop - 10;
cc();
}

function cc() {
    check()
    r_width = eval(document.body.clientWidth) - 120;

    ap01.style.pixelLeft=ap01.style.pixelLeft + 10;

    if (ap01.style.pixelLeft >= r_width)
    {
        clearTimeout(s1);dd();
    }
    else {
        s1 = setTimeout("cc()",100);
    }
}

function dd() {
    check()
    ap01.style.pixelLeft=ap01.style.pixelLeft - 10;

    if (ap01.style.pixelLeft <= 10) {
        clearTimeout(s2);cc();
    }
    else {
        s2 = setTimeout("dd()",100);
    }
}

function fly() {
    var e=event.keyCode;
    switch (e) {
        case 37:
            st01.style.pixelLeft=st01.style.pixelLeft - 10;
            break;
        case 39:
            st01.style.pixelLeft=st01.style.pixelLeft + 10;
            break;
        case 38:
            st01.style.pixelTop=st01.style.pixelTop - 10;
            break;
        case 40:
            st01.style.pixelTop=st01.style.pixelTop + 10;
            break;
        case 83:
            b01.style.display='inline';
            b01.style.pixelLeft=st01.style.pixelLeft + 35;
            b01.style.pixelTop=st01.style.pixelTop - 10;
            shoot();
            break;
    }
}

function shoot() {
    if (b01.style.pixelTop == 0) {

```

```

    b01.style.display='none';
    clearTimeout(sh);
}

else {
    b01.style.top=b01.style.pixelTop - 5;
}
sh = setTimeout("shoot()",5);
}

function check() {
    if ((b01.style.pixelLeft >= ap01.style.pixelLeft) &&
        (b01.style.pixelLeft <= ap01.style.pixelLeft + 70) &&
        (b01.style.pixelTop <= ap01.style.pixelTop))
    {
        ap01.src="explode.gif";
        setTimeout("ap01.style.display='none'",1000);
    }
}

</script>

<body bgcolor=000000; onkeydown="fly();" onLoad=init(>



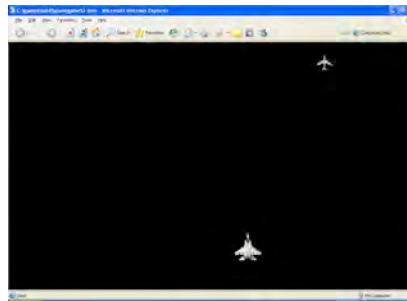


<b id=b01 style="position:absolute; color:white; display:none; z-index=2">!</b>
</body>

</html>

```

3. Test the program. A sample output looks:



### Learning Activity #5: Shooting Crabs

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab8\_5.htm with the following contents:

```

<html>
<style>
.bee {position:relative; background-color:yellow; z-index: 1}
</style>

<script>
function init() {
codes = "";
for (i=1; i<=5; i++) {
    codes += "<img class='bee' id=c" + i + " src='crab.gif' />";
}
}

```

```

    areal.innerHTML += codes;
    moveLeft();
}

function moveLeft() {
    leftSide = areal.style.pixelLeft + 5*32;

    if (leftSide >= 300) {
        if (areal.style.pixelTop >= 200) {
            clearTimeout(sf);
        }
        else {
            areal.style.pixelTop += 20;
            clearTimeout(sf); moveRight();
        }
    }
    else {
        areal.style.pixelLeft += 5;
        sf = setTimeout("moveLeft()", 100);
    }
}

function moveRight() {
    if (areal.style.pixelLeft == 0) {
        if (areal.style.pixelTop >= 200) {
            clearTimeout(sr);
        }
        else {
            areal.style.pixelTop += 20;
            clearTimeout(sr); moveLeft();
        }
    }
    else {
        areal.style.pixelLeft -= 5;
        sr = setTimeout("moveRight()", 100);
    }
}

function shoot() {
    var e=event.keyCode;
    switch(e) {
        case 37:
            shooter.style.pixelLeft-=2; break;
        case 39:
            shooter.style.pixelLeft+=2; break;
        case 83:
            b1.style.pixelLeft = shooter.style.pixelLeft + 14;
            b1.style.display = 'inline';
            fire();
        }
    }
}

function fire() {
    if (b1.style.pixelTop == 0) {
        b1.style.display='none';
        b1.style.top = 270;
        clearTimeout(ss);

    }
    else {
        b1.style.pixelTop -= 5;
        if ((b1.style.pixelTop <= areal.style.pixelTop + 20) &&
            (b1.style.pixelTop >=areal.style.pixelTop)) {

```

```

    if ((b1.style.pixelLeft >= areal.style.pixelLeft) &&
        (b1.style.pixelLeft <= areal.style.pixelLeft + 31)) {
        c1.style.visibility='hidden'; b1.style.display='none';}

    if ((b1.style.pixelLeft >= areal.style.pixelLeft + 32) &&
        (b1.style.pixelLeft <= areal.style.pixelLeft + 61)) {
        c2.style.visibility='hidden'; b1.style.display='none';}

    if ((b1.style.pixelLeft >= areal.style.pixelLeft + 64) &&
        (b1.style.pixelLeft <= areal.style.pixelLeft + 93)) {
        c3.style.visibility='hidden'; b1.style.display='none';}

    if ((b1.style.pixelLeft >= areal.style.pixelLeft + 96) &&
        (b1.style.pixelLeft <= areal.style.pixelLeft + 127)) {
        c4.style.visibility='hidden'; b1.style.display='none';}

    if ((b1.style.pixelLeft >= areal.style.pixelLeft + 128) &&
        (b1.style.pixelLeft <= areal.style.pixelLeft + 159)) {
        c5.style.visibility='hidden'; b1.style.display='none';}
    }
    ss = setTimeout("fire()", 20);
}
}

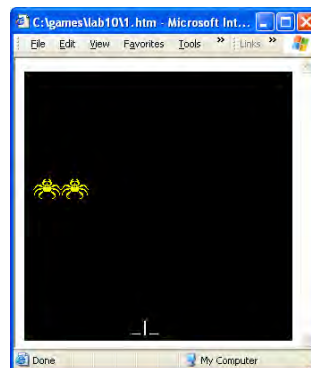
</script>

<body onLoad=init() onKeyDown=shoot()>
<div style="position:absolute; width:300; height:300; background-color: black">
<span id="areal" style="position:absolute;"></span>
<span id="b1" style="position:absolute; top:270;
display:none; color:white; z-index:2">!</span>
<span id="shooter" style="position:absolute; top:275;
color:white"><b>_</b></span>

</div>
</body>
</html>

```

3. Test the program. Press S to shoot, Press → or ← to move to left or right. A sample output looks:



### Submittal

Upon completing all the learning activities,

1. Upload all files you created in this lab to your remote web server.
2. Log in to Blackboard, launch Assignment 08, and then scroll down to question 11.
3. Copy and paste the URLs to the textbox. For example,

- [http://www.geocities.com/cis261/lab8\\_1.htm](http://www.geocities.com/cis261/lab8_1.htm)
- [http://www.geocities.com/cis261/lab8\\_2.htm](http://www.geocities.com/cis261/lab8_2.htm)
- [http://www.geocities.com/cis261/lab8\\_3.htm](http://www.geocities.com/cis261/lab8_3.htm)
- [http://www.geocities.com/cis261/lab8\\_4.htm](http://www.geocities.com/cis261/lab8_4.htm)
- [http://www.geocities.com/cis261/lab8\\_5.htm](http://www.geocities.com/cis261/lab8_5.htm)

No credit is given to broken link(s).

## Game Programming

### Lecture #9 Applying sound effects

**Introduction** In our environment, all the sound you can hear through your ears are analog sounds. Even the sounds that come from the speaker of your computer are analog sounds. However, computers are digital machines, which mean computers do not process analog sounds.

The concept is when a microphone converts sound waves to voltage signals, the resulting signal is an analog (or continuous) signal. The sound card in your computer converts this analog signal to a digital signal for a computer to process.

Analog to digital (A/D) converters handle the task of converting analog signals to digital signals, which is also referred to as sampling. The process of converting an analog signal to a digital signal doesn't always yield exact results. How closely a digital wave matches its analog counterpart is determined by the frequency at which it is sampled, as well as the amount of information stored at each sample.

**Common music file types** Operating systems, such as Windows XP, take care of the communication between the game codes and A/D converter of the sound card. Digital sounds in Windows are known as **waves**, which refer to the physical sound waves from which digital sounds originate. Windows waves are stored in files with a **.WAV** file extension, and can be stored in a wide range of formats to accommodate different sound qualities. More specifically, you can save waves in frequencies from 8kHz to 44kHz with either 8 or 16 bits per sample and as either mono or stereo.

Musical Instrument Digital Interface, or **MIDI**, has established a standard interface between musical instruments. MIDI is commonly used with a dedicated keyboard to control a synthesizer. The keyboard inputs musical information (such as musical notes), such that the synthesizer can generate electronic sound output (such as songs). MIDI is popular in generating sound effects for games.

**Creating and editing sounds** To be able to create and modify your own sounds, you need some type of software sound editing utility. Sound editing utilities usually provide a means of sampling sounds from a microphone, CD-ROM, or line input. There are many free, open source, sound editing tools you can easily find on the Internet.

The truth is that no matter what tool you choose to use, the process of editing sounds is technically the same.

- **waveform**: Determine whether the sound volume is too loud or too soft. Waveform of a sound is the graphical appearance of the sound when plotted over time.
- **Clipping**: zooming in on the waveform in a sound editor and cutting out the silence that appears before and after the sound.
- **Latency** is the amount of time between when you queue a sound for playing and when the user actually hears it. Latency should be kept to a minimum so that sounds are heard when you want them to be heard without any delay.

**Background sound/music** DHTML uses HTML's **<BGSOUND>** and **<EMBED>** tags to allow you to associate a sound file to a web game page as background sounds or soundtracks. Although it plays a background sound when the page is opened, you then uses the **autostart="false"** property and value to keep the sound file from being played automatically.

The **<BGSOUND>** tag may be placed either within the **<HEAD>..</HEAD>** or the **<BODY>..</BODY>** portion of the page. For example,

```
<HTML><BODY>
```

```
<bgsound SRC="game1.mid" autostart="true" />
</BODY></HTML>
```

and

```
<HTML><head>
<bgsound SRC="game1.mid" autostart="false" />
</head></HTML>
```

Microsoft documents specify more parameters for the <BGSOUND> tag than are currently supported. The following table lists the supported ones:

Parameter	Description
autostart="value"	Set the value to TRUE to being playing the music immediately upon page load.
ID="idvalue"	An identifier to be used for references in associated style sheet, hypertext links, and JavaScript code. Must begin with a letter and may include underscores. ID should be unique within the scope of the document. If duplicated, the collection of identical-ID elements may be referenced by their ordinal numbers.
LOOP="n"	Specifies how many times the sound will loop.
SRC="url"	Specifies the URL of the sound file. As shown above, sound files can be in any recognizable format (for example: "midi, wav, au")
VOLUME="n"	Determines the loudness of the background sound. Values may range from -10,000 (weakest) to 0 (loudest). Not supported by the MAC.

You can use the HTML **TITLE** attribute to describe an HTML tag. This capability is particularly important for elements that do not normally have textual information associated with them, such as the **bgSound** object, which specifies a sound file to play in the background of a Web page.

The following example uses the TITLE attribute with the bgSound object to describe the background sound.

```
<BGSOUND SRC="soundfile.mid" TITLE="Sound of falling water" />
```

The <**EMBED**> tag allows documents of any type to be embedded, including music files. For example, the following use of the EMBED element mimics the behavior of the BGSOUND tag.

```
<EMBED type="audio/x-midi" src="cis261.mid" autostart="false"
hidden="true">
```

When you embed a sound, it plays as soon as the page loads, you can use the **autostart** = "false" property and values to prevent the sound file from being played as soon as the page is loaded.

The **hidden** property, when being set to true, controls the visibility of the sound file icon. If you do not supply HIDDEN, then the sound file icon will be visible, as shown below.



The EMBED element must appear inside the BODY element of the document. Additionally, Internet Explorer relies on the resource specified in the SRC attribute to ultimately determine which application to use to display the embedded document.

You can call a sound playback function from the onMouseOver event handler. For example,

```

<button
onMouseOver="playSound();return true"
onMouseOut ="stopSound();return true"
>Play</button>

```

This technique applies to the design of the Piano game. When a user move mouse over the key, the computer plays the Csound file.

```

.....
.....
<span class="wKey" id="midC" style="left:50"
onMouseOver="Down();document.Csound.play()"
onMouseOut="Up()" "></span>
.....
.....
<span class="bKey" id="cSharp" style="left:72"
onMouseOver="document.Cs.play()" "></span>
.....
.....
<embed src="C0.wav" autostart=false hidden=true name="Csound"
mastersound>
.....
.....
</html>

```

You can add an sound effect to the Jet Fighter game (you created it in a previous lecture) by adding the following bold faced lines:

```

function check() {
  if ((b01.style.pixelLeft >= ap01.style.pixelLeft) &&
      (b01.style.pixelLeft <= ap01.style.pixelLeft + 70) &&
      (b01.style.pixelTop <= ap01.style.pixelTop + 70))
  {
    document.explosion.play()
    ap01.src="explode.gif";
    setTimeout("ap01.style.display='none'",1000);
  }
}

</script>

<body bgcolor=000000; onkeydown="fly();" onLoad=init(>


<b id=b01 style="position:absolute; color:white; display:none; z-
index=2">!</b>

<embed src="explosion.wav" autostart=false hidden=true
name="explosion" mastersound>

</body>

```

Certainly, you must load the explosion.wav sound file to the same directory as the Jet Fighter file resides.

Play sound  
/music file  
using  
JavaScript

With Microsoft Internet Explorer, there are several ways to play a sound file using DHTML and JavaScript, without opening a separate window for sound control. Assuming you add the following MIDI file:

```

<embed name="mysound" src="game1.mid"

```

```
autostart=false hidden=true mastersound>
```

To start playing the `game1.mid` file via the EMBED tag, use:

```
document.mySound.play()
```

To stop playing, use:

```
document.mySound.stop()
```

A complete code sample is:

```
<script>
function playSound(){
    document.sound1.play();
}

function stopSound(){
    document.sound1.stop();
}
</script>
<body>
<embed src="game1.mid" autostart="false" hidden="True"
name="sound1">

<button onClick="playSound()">Play</button>
<button onClick="stopSound()">Stop</button>
</body>
```

If you use the `<bgsound>` tag, you have to do it differently. Assuming you add the sound file using the following code:

```
<bgsound id="kick" loop=1>
```

To start playing a sound file `game1.mid` via the BGSOUND tag, use:

```
document.all['kick'].src='game1.mid'
```

To stop playing, use:

```
document.all['kick'].src='nosound.mid'
```

Notice that **game1.mid** stands for the name of the sound file that you actually want to play; **nosound.mid** is a “do-nothing” sound file, which means it does not play any sound at all, but can be used to stop the playback of other sound files.

A complete code sample is:

```
<script>
function playSound(){
    document.all['kick'].src='game1.mid'
}

function stopSound(){
    document.all['kick'].src='nosound.mid'
}
</script>

<button onClick="playSound()">Play</button>
<button onClick="stopSound()">Stop</button>
```

```
<bgsound id="kick" loop=1 autostart="false">
```

One trick you need to know is that you don't need to use the **<bgsound>** tag to include all files you need to play, simply include the first sound file is good enough. Remember, the more sound files you include to your code using **<bgsound>** and **<embed>** tags, the longer it takes to load the game file. For example, you can use JavaScript to switch between the game1.mid and game2.mid soundtracks every 20 seconds.

```
<BODY>
<BGSOUND SRC="game1.mid" ID="cis261" autostart="true" />
<SCRIPT>
setInterval("changeSound()",20000);

function changeSound(){
    if (document.all.cis261.src == "game1.mid")
        document.all.cis261.src = "game2.mid"
    else document.all.cis261.src = "game1.mid";
}

</SCRIPT>
</BODY>
```

#### JavaScript Sound effect

Want to add a short sound effect to your page for certain actions, such as when the user moves the mouse over a link? This is a simple yet versatile script that lets you do just like! Relying on IE's BGSOUND attribute (and hence IE5+ only), the script can easily add a sound effect to a single item (ie: 1 link), or thanks to a helper function, all items of the specified element (ie: all <a> tags). This makes it very easy to add a sound effect to an entire menu's links, for example.

Add the below script to the <HEAD> section of your page, and change "var soundfile" to point to where your sound file is located.

```
<bgsound src="#" id="soundeffect" loop=1 autostart="true" />

<script>

var soundfile="game1.mid"

function playsound(soundfile){
    if (document.all && document.getElementById){
        document.getElementById("soundeffect").src="" //reset first in
case of problems
        document.getElementById("soundeffect").src=soundfile
    }
}

function bindsound(tag, soundfile, masterElement){
    if (!window.event) return
    var source=event.srcElement
    while (source!=masterElement && source.tagName!="HTML"){
        if (source.tagName==tag.toUpperCase()){
            playsound(soundfile)
            break
        }
        source=source.parentElement
    }
}

</script>
```

Set up an element to receive the JavaScript sound, whether onMouseover, onClick etc. For example, the below plays a sound when the mouse moves over a link:

```
<button onClick="playsound(soundfile)">Play 1</button>  
or  
<button onMouseover="playsound(soundfile)">Play 2</button>
```

The second line shows that you can pass in a different wav file to play for any link other than the default specified within the script (var soundfile).

## Digital sound theory

To sample a sound, you just store the amplitude of the sound wave at regular intervals. Taking samples at more frequent intervals causes the digital signal to more closely approximate the analog signal and, therefore, sound more like the analog wave when played. So, when sampling sounds the rate (frequency) at which the sound is sampled is very important, as well as how much data is stored for each sample. The unit of measurement for frequency is Hertz (Hz), which specifies how many samples are taken per second. As an example, CD-quality audio is sampled at 44,000Hz (44kHz), which means that when you're listening to a music CD you're actually hearing 44,000 digital sound samples every second.

In addition to the frequency of a sampled sound, the number of bits used to represent the amplitude of the sound impacts the sound quality, as well as whether the sound is a stereo or mono sound. Knowing this, it's possible to categorize the quality of a digital sound according to the following properties:

- Frequency
- Bits-per-sample
- Mono/stereo

The frequency of a sampled sound typically falls somewhere in the range of 8kHz to 44kHz, with 44kHz representing CD-quality sound. The bits-per-sample of a sound is usually either 8bps (bits per sample) or 16bps, with 16bps representing CD-quality audio; this is also known as 16-bit audio. A sampled sound is then classified as being either mono or stereo, with mono meaning that there is only one channel of sound, whereas stereo has two channels. As you might expect, a stereo sound contains twice as much information as a mono sound. Not surprisingly, CD-quality audio is always stereo. Therefore, you now understand that a CD-quality sound is a 44kHz 16-bit stereo sound.

Although it would be great to incorporate sounds that are CD-quality into all of your games, the reality is that high-quality sounds take up a lot of memory and can therefore be burdensome to play if your game already relies on a lot of images and other memory-intensive resources. Granted, most computers these days are capable of ripping through memory-intensive multimedia objects such as MP3 songs like they are nothing, but games must be designed for extreme efficiency. Therefore, it's important to consider ways to minimize the memory and processing burden on games every chance you get. One way to minimize this burden is to carefully choose a sound quality that sounds good without hogging too much memory.

## Review Questions

1. Where can you place a sound file using the <BGSOUND> tag?  
A. between <head> and </head>  
B. between <body> and </body>  
C. A and B  
D. None of the above
2. Where can you place a sound file using the <EMBED> tag?  
A. between <head> and </head>  
B. between <body> and </body>  
C. A and B  
D. None of the above
3. Which causes the cis261.mid sound file to play automatically when the page is loaded?  
A. <bgsound src="cis261.mid" autostart="true">

- B. <bgsound src="cis261.mid" autostart="yes">
- C. <bgsound src="cis261.mid" start=1>
- D. <bgsound src="cis261.mid" start="auto">

4. Which sets the embedded MIDI file as an visible object on the page?

- A. <EMBED src="cis261.mid" display="true">
- B. <EMBED src="cis261.mid" hidden="true">
- C. <EMBED src="cis261.mid" hidden="visible">
- D. <EMBED src="cis261.mid" display="invisible">

5. Given the following code segment, which can play the sound?

```
<embed name="s1" src="cis261.mid" autostart="false" mastersound>
```

- A. window.s1.play();
- B. document.s1.play();
- C. window.play(s1);
- D. document.play(s1);

6. Given the following code segment, which can play the sound?

```
<bgsound id="c1" autostart="false">
```

- A. document.all['c1'].src='c261.mid'
- B. document.play('c1'); c1.src='c261.mid'
- C. document.all.c1.src == "c261.mid"
- D. document.play('c1').src='c261.mid'

7. Given the following code segment, which is a "do-nothing" sound file?

```
function playSound() {
    var b1="m2.mid";
    document.all['a1'].src='m1.mid';
    document.b1.play(); }

function stopSound() {
    var d1="m4.mid";
    document.all['c1'].src='m3.mid';
    document.d1.stop(); }
</script>
```

- A. m1.mid
- B. m2.mid
- C. m3.mid
- D. m4.mid

8. When click the following button, which sound file will be played?

```
<button onMouseOver="Down();document.E.play()">Play</button>
```

- A. <embed src="C.wav" autostart=false hidden=true name="C" mastersound>
- B. <embed src="D.wav" autostart=false hidden=true name="D" mastersound>
- C. <embed src="E.wav" autostart=false hidden=true name="E" mastersound>
- D. <embed src="F.wav" autostart=false hidden=true name="F" mastersound>

9. Given the following code segment, which sound will be played automatically?

```
<embed id="s1" src="hit.wav" autostart=false mastersound>
<bgsound id="s2" src="circusride.mid">
```

`<bgsound id="s3" />`

- A. s1
- B. s2
- C. s3
- D. All of the above

10. If you wish to switch between two MIDI files every 20 seconds, which method will you choose to use?

- A. setInterval()
- B. setLoop()
- C. setSwitch()
- D. setReturn()

## Game Programming

### Lab #9

#### Preparation #1:

1. Create a new directory named **C:\games**.
2. Use Internet Explorer to go to **http://business.cypresscollege.edu/~pwu/cis261/download.htm** to download lab9.zip (a zipped) file. Extract the files to C:\games directory.

#### Learning Activity #1:

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Rename the **jet\_fighter.htm** file to **lab9\_1.htm**.
3. Use Notepad to open the **C:\games\lab9\_1.htm** file.
4. Add the following bold-faced lines:

```
.....
.....
function check() {
    if ((b01.style.pixelLeft >= ap01.style.pixelLeft) &&
        (b01.style.pixelLeft <= ap01.style.pixelLeft + 70) &&
        (b01.style.pixelTop <= ap01.style.pixelTop + 70))
    {
        document.explosion.play()
        ap01.src="explode.gif";
        setTimeout("ap01.style.display='none'",1000);
    }
}

</script>

<body bgcolor=000000; onkeydown="fly();" onLoad=init()>


<b id=b01 style="position:absolute; color:white; display:none; z-index=2">!</b>

<embed src="explosion.wav" autostart=false hidden=true name="explosion"
mastersound>

</body>
</html>
```

5. Test the program. Whenever you hit the enemy's plan, the explosion.wav sound file is played.

#### Learning Activity #2: Gun shot

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named **C:\games\lab9\_2.htm** with the following contents:

```
<html>

<script>
```

```

function shoot() {
var i = b01.style.pixelLeft;

if (i <= 400) {
  b01.style.display='inline';
  b01.style.left=i + 10;
  setTimeout("shoot()",20);
}
else {
  b01.style.display='none';
  b01.style.left=95;
};
}

function playsound() {
document.all['gunshot'].src='gunshot.wav'
}

</script>

<body onKeyDown="playsound();shoot();">

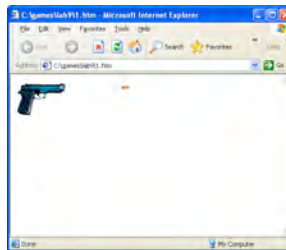

</span>

<bgsound id="gunshot" />

</body>
</html>

```

3. Test the program. Press any key to fire the gunshot; you should hear the sound effect, too. A sample output looks:



### Learning Activity #3: Squeeze the clown

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab9\_3.htm with the following contents:

```

<html>

<script>
var k=0;

function goUp() {
  bar.style.display='inline';
  bar.style.top=bar.style.pixelTop - 5;

```

```

if (clown.style.pixelTop <= 20) {
    clown.style.height=eval(bar.style.top.replace('px','')) - 20;
    clown.style.width = eval(clown.style.width.replace('px',''))+1;
    clown.style.Top = 20;
    document.hit.play()
}
else {
    bar.style.height=k;
    k+=5;
    clown.style.top=bar.style.pixelTop-120;
}

if ( eval(clown.style.height.replace('px','')) <=10) {
    clearTimeout(s1);}
else {
    s1=setTimeout("goUp()",20); }
}

</script>

<body onKeyDown=goUp()>
<hr style="position:absolute; left:10; top:0;
background-color:red; width:100;z-index:2" size="20px" align="left">

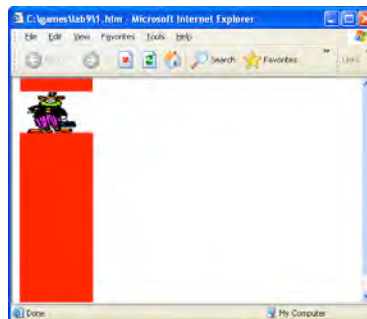


<span id="bar" style="position:absolute; left:10; top:400;
background-color:red; width:100; display:none; z-index:1"></span>

<embed src="hit.wav" autostart=false hidden=true name="hit" mastersound>
<bgsound src="circusride.mid">
</body>
</html>

```

3. Test the program. When the page is loaded, the background music plays. When squeezing the clown, you will hear a sound effect. A sample output looks:



#### Learning Activity #4

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab9\_4.htm with the following contents:

```

<html>
<script>
var x=190; y=384;

```

```

var k=1; n=0;
function draw(){
var i = event.keyCode;
  switch (i) {
    case 37: n=1; k=0; document.left.play(); break;
    case 38: n=0; k=1; document.up.play(); break;
    case 39: n=-1; k=0; document.right.play(); break;
    case 40: n=0; k=-1; document.down.play(); break;
  }
}

function cc() {

if ((x<=10) || (x>=394) || (y<=-10) || (y>=400)) {
pl.innerText="Game over!";
}
else {
y-=k; x-=n;
codes = "<span style='position:absolute;left:"+x;
codes += "; top:"+ y +"'.</span>";
areal.innerHTML += codes;
setTimeout("cc()", 0.1);
}
}

</script>

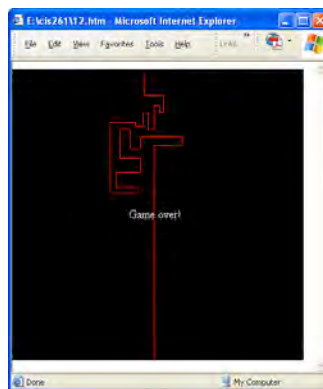
<body OnLoad=cc() onKeyDown=draw()>
<div id=areal style="position:absolute; width:400;
  height:400; border:solid 3 black; color:red;
  background-color: black; left:0; top:10;"></div>

<p id=p1 style="position:absolute; left:160;top:200; color: white"></p>

<embed src="left.wav" autostart=false hidden=true name="left" mastersound>
<embed src="right.wav" autostart=false hidden=true name="right" mastersound>
<embed src="up.wav" autostart=false hidden=true name="up" mastersound>
<embed src="down.wav" autostart=false hidden=true name="down" mastersound>
<bgsound src="canzone.mid">
</body>
</html>

```

3. Test the program. A sample output looks:



## Learning Activity #5: Piano

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab9\_5.htm with the following contents:

```
<html>
<style>
.wKey {
    position:absolute;
    top:20;
    background-color:white;
    border-top:solid 1 #cdcdcd;
    border-left:solid 1 #cdcdcd;
    border-bottom:solid 4 #cdcdcd;
    border-right:solid 1 black;
    height:150px;
    width:40px
}

.bKey {
    position:absolute;
    top:20;
    background-color:black;
    border:solid 1 white;
    height:70px;
    width:36px
}
</style>

<script>
function Down() {
var keyID = event.srcElement.id;
document.getElementById(keyID).style.borderTop="solid 1 black";
document.getElementById(keyID).style.borderLeft="solid 1 black";
document.getElementById(keyID).style.borderBottom="solid 1 black";
document.getElementById(keyID).style.borderRight="solid 1 #cdcdcd";
}

function Up() {
var keyID = event.srcElement.id;
document.getElementById(keyID).style.borderTop="solid 1 #cdcdcd";
document.getElementById(keyID).style.borderLeft="solid 1 #cdcdcd";
document.getElementById(keyID).style.borderBottom="solid 4 #cdcdcd";
document.getElementById(keyID).style.borderRight="solid 1 black";
}
</script>

<div>

<span class="wKey" id="midC" style="left:50"
    onMouseOver="Down();document.Csound.play()"
    onMouseOut="Up()" "></span>

<span class="wKey" id="midD" style="left:91"
    onMouseOver="Down();document.Dsound.play()"
    onMouseOut="Up()" "></span>

<span class="wKey" id="midE" style="left:132"
    onMouseOver="Down();document.Esound.play()"
    onMouseOut="Up()" "></span>
```

```

<span class="wKey" id="midF" style="left:173"
  onMouseOver="Down();document.Fsound.play()"
  onMouseOut="Up()" "></span>

<span class="wKey" id="midG" style="left:214"
  onMouseOver="Down();document.Gsound.play()"
  onMouseOut="Up()" "></span>

<span class="wKey" id="midA" style="left:255"
  onMouseOver="Down();document.Asound.play()"
  onMouseOut="Up()" "></span>

<span class="wKey" id="midB" style="left:296"
  onMouseOver="Down();document.Bsound.play()"
  onMouseOut="Up()" "></span>

<span class="wKey" id="highC" style="left:337"
  onMouseOver="Down();document.CHsound.play()"
  onMouseOut="Up()" "></span>

<span class="bKey" id="cSharp" style="left:72"
onMouseOver="document.Cs.play()" "></span>
<span class="bKey" id="dSharp" style="left:114"
onMouseOver="document.Ds.play()" "></span>
<span class="bKey" id="fSharp" style="left:196"
onMouseOver="document.Fs.play()" "></span>
<span class="bKey" id="gSharp" style="left:238"
onMouseOver="document.Fs.play()" "></span>
<span class="bKey" id="aSharp" style="left:280"
onMouseOver="document.As.play()" "></span>
</div>

<!-- white key sound files -->
<embed src="C0.wav" autostart=false hidden=true name="Csound" mastersound>
<embed src="D0.wav" autostart=false hidden=true name="Dsound" mastersound>
<embed src="E0.wav" autostart=false hidden=true name="Esound" mastersound>
<embed src="F0.wav" autostart=false hidden=true name="Fsound" mastersound>
<embed src="G0.wav" autostart=false hidden=true name="Gsound" mastersound>
<embed src="A0.wav" autostart=false hidden=true name="Asound" mastersound>
<embed src="B0.wav" autostart=false hidden=true name="Bsound" mastersound>
<embed src="C1.wav" autostart=false hidden=true name="CHsound" mastersound>

<!-- black key sound files -->
<embed src="Cs0.wav" autostart=false hidden=true name="Cs" mastersound>
<embed src="Ds0.wav" autostart=false hidden=true name="Ds" mastersound>
<embed src="Fs0.wav" autostart=false hidden=true name="Fs" mastersound>
<embed src="Gs0.wav" autostart=false hidden=true name="Gs" mastersound>
<embed src="As0.wav" autostart=false hidden=true name="As" mastersound>
</html>

```

3. Test the program. Use the mouse to play a piece of music now.



**Submittal**

Upon completing all the learning activities,

1. Upload all files you created in this lab to your remote web server.
2. Log in to Blackboard, launch Assignment 09, and then scroll down to question 11.
3. Copy and paste the URLs to the textbox. For example,
  - [http://www.geocities.com/cis261/lab9\\_1.htm](http://www.geocities.com/cis261/lab9_1.htm)
  - [http://www.geocities.com/cis261/lab9\\_2.htm](http://www.geocities.com/cis261/lab9_2.htm)
  - [http://www.geocities.com/cis261/lab9\\_3.htm](http://www.geocities.com/cis261/lab9_3.htm)
  - [http://www.geocities.com/cis261/lab9\\_4.htm](http://www.geocities.com/cis261/lab9_4.htm)
  - [http://www.geocities.com/cis261/lab9\\_5.htm](http://www.geocities.com/cis261/lab9_5.htm)

No credit is given to broken link(s).

## Game Programming

### Lecture #10      Score boards

**Introduction**      Many people consider scoring the extension of collision detection, and should be part of the collision response. Once you create a game code that allows two or more objects to collide, you frequently need a method to keep and display the score the play earned. However, there are many situations in which you have nothing to collide when keeping the game score. The instructor, thus, believes score keeping should be an individual topic, and is completely independent of the collision response.

The scoring keeping function can be an individual function, or it can be placed into other function. This lecture will introduce you to the world of scoring in game programming.

**Use variables to keep the score**      In a computer game, the player earns or loses score points in according to the game rules the programmers defined. The score points may increase, decrease, or stay the same depending on how the players stick to the game rules. In other words, the score point is not a fixed value, because it varies as the players continue to play the game.

In terms of programming, programmers declare variables to keep such continuously changing values in a temporary holding area (namely computer's memory). If the values need to be updated, the new value replaces the old one to be stored in that temporary holding area. It is necessary to assign a name to that holding area, such that the computer can always locate the correct temporary holding area if there are more than one variable.

In any programming language, the process to tell the computer to find some memory space as a temporary particular holding area is known as “declaring a variable.” The name of the variable is also the ID of the temporary holding area. The values of the variable, such as a string, a number, or a date which directly represents a constant value, are known as **literals**.

In JavaScript, you declare a variable by using the following syntax:

```
var VariableName = "InitialValue";
```

where **var** is a keyword, *VariableName* is the name you assign to that variable. If the variable has an initial value (meaning its first value before being updated by the computer), add it next to the = sign. For example,

```
var playerName = "Helen";  
var gameName = 'Tic Tac Toe';  
var score = 0;
```

The use of single (') or double quotes (") specifies that these two values--**Helen** and **Tic Tac Toe**--are strings. The term “string” refers to a series of alphanumeric characters, usually forming a piece of text.

A game code segment consists of lines of statements with sequence arranged by the programmer. When computer execute the codes, statements are executed on a sequential basis. Consider the following code segments:

```
var myScore = 0;  
myScore = myScore + 5;  
myScore += 50;  
myScore = myScore - 10;  
myScore -= 5;
```

The first line declares the variable **myScore** with an initial value **0**. The second line means “myScore now equals to whatever myScore currently has plus 5”, so the new value is 5 after this statement is executed ( $0 + 5 = 5$ ).

The following line uses the increment operator (**+=**) to tell the computer to “add 50 to the current value of myScore”, so the value after execution is 55 ( $5 + 50 = 55$ ).

```
myScore += 50;
```

The increment (**++**) operator is a unary operator that can increment a value by adding 1 to it. You can either place the increment operator to the left of a variable (known as pre-increment), or place it to the right of a variable (known as post-increment). In the following example, both x and y variables will have a new value 1 after the execution.

```
var x=0; y=0;
++x;
y++;
```

When you need to increment by adding 2 or larger, the increment operator changes to (**+=n**, where *n* is usually an integer). For example,

```
myScore += 50;
```

The following line tells the computer to subtract 10 from the current value of myScore, so the value after execution is 45 ( $55 - 10 = 45$ ).

```
myScore = myScore - 10;
```

The last line uses the decrement operator (**-=**) to subtract 5 from the current value of myScore, so the value after execution is 40 ( $45 - 5 = 40$ ).

The concept of decrement (**--**) operator is opposite to that of increment operator. So, the values of x and y after execution in the following example are 0.

```
var x=1; y=1;
x--;
y--;
```

When you need to decrement by 2 or larger, the operator changes to (**-=n**, where *n* is usually an integer).

Adding /  
Decreasing  
Score:

Keeping score in essence is to add or subtract certain number from a variable’s current value. In the number guessing game, the following code creates two textboxes: one (t1) is visible; the other (t2) is invisible.

The **start()** function use the **Math.random()** method to generate a random number in the range of [1-64] (or from 1 to 64). The random number is then temporarily assigned to the **t2** textbox as its value. The **t1** textbox, on the other hand, is for the player to enter a number to guess what number the computer will randomly generates.

```
.....
.....
function start() {
var k=Math.floor(Math.random()*64);
f1.t2.value=k;
.....
.....
<form name=f1>
```

```

Enter a number (1-64):


```

When the player clicks the Stop button to trigger the stop() function, the **score** variable is used to update and keep the increment (adding) or decrement (decreasing) of the player's score.

```

function stop() {
clearTimeout(sf);
if (f1.t1.value==f1.t2.value) { score+=1000; p1.innerText = score;}
else { score -=1; p1.innerText = score;}
}

```

The logic is simple. Compare the value of the player's entry, t1, and the random number generated by the computer (which is temporarily stored to t2 textbox). If the are exactly the same, the **score** variable increments it value of 1000; otherwise, it decrement by 1.

The following line displays the updated score in the **p1** object, the **innerText** property forces the value to be a **string** (text-based) literal.

```

p1.innerText = score;

```

In the Punching Duck game, the punch() function forces the glove to move toward northeast till the **gv.style.pixelTop** value is less than or equal to 0.

```

function punch() {
gv.style.pixelLeft+=2;
gv.style.pixelTop-=2;

if (gv.style.pixelTop <=0)
{clearTimeout(s2);gv.style.pixelTop=200;gv.style.pixelLeft=10;}
else {s2=setTimeout("punch()", 20);}

for (i=1; i<=10; i++) {
code3="if ((gv.style.pixelLeft >= d"+i+".style.pixelLeft) &&";
code3+=" (gv.style.pixelLeft <= (d"+i+".style.pixelLeft + 61)) && ";
code3+=" (gv.style.pixelTop <= d"+i+".style.pixelTop)) ";
code3+="{d"+i+".style.display='none';d"+i+".src='noduck.gif';";
code3+="score+=10;p1.innerText=score}";
eval(code3);
}
}

```

When the glove moves, the following code block generates 10 similar codes segments which will detect the collision of the **gv** object with any of the 10 ducks (d1, d2, d3, ..., d10).

```

for (i=1; i<=10; i++) {
code3="if ((gv.style.pixelLeft >= d"+i+".style.pixelLeft) &&";
code3+=" (gv.style.pixelLeft <= (d"+i+".style.pixelLeft + 61)) && ";
code3+=" (gv.style.pixelTop <= d"+i+".style.pixelTop)) ";
code3+="{d"+i+".style.display='none';d"+i+".src='noduck.gif';";
code3+="score+=10;p1.innerText=score}";
eval(code3);
}

```

The following is a sample output of the above code block for d3:

```
if ((gv.style.pixelLeft >= d3.style.pixelLeft) &&
    (gv.style.pixelLeft <= (d3.style.pixelLeft + 61)) &&
    (gv.style.pixelTop <= d3.style.pixelTop))
{d3.style.display='none';d3.src='noduck.gif'
  score+=10;p1.innerText=score }
```

In plain English, it means “if gv’s pixelLeft value is greater than or equal to that of d3 and gv’s pixelLeft value is less than or equal to d3’s plus 61 (the width of d3) and gv’s pixelTop value is less than or equal to d3’s, then .... add 10 to the value of the variable **score**.....”

In other words, the above *for* loop builds a system of collision detection for each of the 10 ducks. The collision detection is based on three borders—left, right, and bottom, of each of the  $d_n$  (where  $n$  is 1, 2, 3, ... 10) object.



Notice that it takes 5 movements for the glove to move across the duck, as shown below:



Each movement forces the computer to add 10 to the value of **score**, so the player earns 50 points (50 + 50 + 50 + 50 + 50) each time when he/she punches the duck with ID **d3**.

In the Shooting Crab game, the **fire()** function detects the collision of **c1** and **area1** object. If the collision happens, the score variable is incremented by 10.

```
function fire() {
  if (b1.style.pixelTop == 0) {
    b1.style.display='none';
    b1.style.top = 270;
    clearTimeout(ss);
  }
  else {
    b1.style.pixelTop -= 5;
    if ((b1.style.pixelTop <= area1.style.pixelTop + 20) &&
        (b1.style.pixelTop >=area1.style.pixelTop)) {
      if ((b1.style.pixelLeft >= area1.style.pixelLeft) &&
          (b1.style.pixelLeft <= area1.style.pixelLeft + 31))
      {
        c1.style.display='none';b1.style.display="none";
        clearTimeout(ss); score += 10;
        gd.innerText = score; newCrab() }
      }
    ss = setTimeout("fire()", 20);
  }
}
```

Store the score in a JavaScript	JavaScript can create, read, and erase HTTP (HyperText Transfer Protocol) cookies. A cookie is a small text file a Web server sends to a web client computer to store information about the user. A
---------------------------------	---

## Cookie

JavaScript game may set a cookie file to keep the game score temporarily.

Cookies allow you to store information on the client computer. However, a cookie can only hold string based name/value pairs (i.e *name=value* settings). The cookie property of the document object set the cookie. The format is:

```
document.cookie = "CookieName=Value";
```

Consider the following example. The **scr** variable keeps the updated value.

```
<html>
<script>
var scr = 0;
var d = new Date();
function set_score() {
    document.cookie = "score="+scr;
    scr += 5;
}

function read_score() {
    p1.innerText = "Score: " + document.cookie.split("=")[1];
}
</script>

<body onLoad="set_score();read_score()">

<button onClick="set_score();read_score()">Click</button>
<p id="p1"></p>

</body>
</html>
```

Since the data stored in a cookie has a format of *CookieName=Value*, you can use the `split()` function to break the data into a string array in which *CookieName* is the first element and *Value* is the *second*. The following get the seconds elements:

```
document.cookie.split("=")[1]
```

Similarly, to retrieve the name of cookie, use

```
document.cookie.split("=")[0]
```

## Store the score in a text file

Values held by a variable can then be saved to local file permanently or kept in the computer's physical memory temporarily. JavaScript support reading from and writing to local file with the support of ActiveX objects (Internet Explorer only).

The **OpenTextFile** method opens a specified file and returns a `TextStream` object that can be used to read from, write to, or append to the file. The syntax is:

```
object.OpenTextFile(filename[, iomode[, create[, format]])
```

where,

- *object*: is always the name of a `FileSystemObject`.
- *filename*: String expression that identifies the file to open.
- *iomode*: Can be one of three constants: `ForReading` (1), `ForWriting` (2), or `ForAppending` (8).
- *create*: Boolean value that indicates whether a new file can be created if the specified filename doesn't exist. The value is `True` if a new file is created, `False` if it isn't created. If omitted, a new file isn't created.
- *format*: One of three Tristate values used to indicate the format of the opened file. If omitted,

the file is opened as ASCII.

The **CreateTextFile** method Creates a specified file name and returns a TextStream object that can be used to read from or write to the file. The syntax is:

```
object.CreateTextFile(filename[, overwrite[, unicode]])
```

where,

- *object*: the name of a FileSystemObject or Folder object.
- *filename*: String expression that identifies the file to create.
- *overwrite*: Boolean value that indicates whether you can overwrite an existing file. The value is true if the file can be overwritten, false if it can't be overwritten. If omitted, existing files are not overwritten.
- *unicode*: Boolean value that indicates whether the file is created as a Unicode or ASCII file. The value is true if the file is created as a Unicode file, false if it's created as an ASCII file. If omitted, an ASCII file is assumed.

Consider the following code, which illustrates how to use the CreateTextFile method to create and open a text file.

```
var fso = new ActiveXObject("Scripting.FileSystemObject");  
var a = fso.CreateTextFile("c:\\testfile.txt", true);  
a.WriteLine("This is a test.");  
a.Close();
```

The following code illustrates the use of the OpenTextFile method to open a file for appending text:

```
var fs, a, ForAppending;  
ForAppending = 8;  
fs = new ActiveXObject("Scripting.FileSystemObject");  
a = fs.OpenTextFile("c:\\testfile.txt", ForAppending, false);  
...  
a.Close();
```

You can simplify the above code to:

```
var fs, a;  
fs = new ActiveXObject("Scripting.FileSystemObject");  
a = fs.OpenTextFile("c:\\testfile.txt", 8, false);  
...  
a.Close();
```

Consider the following code.

```
<html><head>  
<SCRIPT LANGUAGE='JavaScript'>  
var scr=0;  
  
function WriteToFile() {  
    var filename = 'c://tmp//score.txt';  
    var fso = new ActiveXObject('Scripting.FileSystemObject');  
    if (fso.FileExists(filename)) {  
        var a, file;  
        file = fso.OpenTextFile(filename, 2, false);  
        file.WriteLine(scr);  
    }  
    else {  
        var file = fso.CreateTextFile(filename, true);  
        file.WriteLine(scr);  
    }  
}
```

```

        file.Close();
        scr+=5;
    }

    function ReadIt() {
        var filename = 'c://tmp//score.txt';

        var fso, a, ForReading;
        fso = new ActiveXObject('Scripting.FileSystemObject');
        file = fso.OpenTextFile(filename, 1, false);
        p1.innerText = file.readline();
        file.Close();
    }
</SCRIPT>
</head>
<body onload='WriteToFile();ReadIt() '>
<button onClick='WriteToFile();ReadIt()'> Click </button>
<p id="p1"></p>
</body>
</html>

```

Keeping score based on game rules

As the game programmer, you have the power of defining what the game rules are. However, you need to keep the score based on the game rule, and you need to absolutely stick to the rules. In the Slot Machine game, for example, the rules are:

- If 4 numbers are all 7's, win 30000 points.
- If there are any 3 numbers having 7 as values, win 3000 points.
- If there are any 3 numbers having exactly the same values, win 300 points.
- If there are any 2 numbers having exactly the same value, assign 0 point.
- If all the 4 numbers are all different, lose 30 points.

The **check()** code is responsible for validating the above game rules and keep score based on the rules.

```

function check() {
    clearTimeout(s1);

    var v1 = f1.n1.value;
    var v2 = f1.n2.value;
    var v3 = f1.n3.value;
    var v4 = f1.n4.value;

    if ((v1 == 7) && (v2 == 7) && (v3 == 7) && (v4==7))
        {gd += 30000; score.innerText = 30000;}

    else if (((v1 == 7) && (v2 == 7) && (v3 == 7)) ||
        ((v1 == 7) && (v2 == 7) && (v4 == 7)) ||
        ((v2 == 7) && (v3 == 7) && (v4 == 7)))
        {gd += 3000; score.innerText = 3000;}

    else if (((v1 == v2) && (v2 == v3)) ||
        ((v1 == v2) && (v2 == v4)) ||
        ((v2 == v3) && (v3 == v4))) { gd += 300;
        score.innerText = 300; }

    else if ((v1 == v2) || (v1 == v3) ||
        (v1 == v4) || (v2 == v3) ||
        (v2 == v4) || (v3 == v4))
        {gd += 0; score.innerText = 0;}
}

```

```

else {gd -=30; score.innerText = "-30";}

credit.innerText = gd;
}
</script>

```

The following line, for example, defines the rule “If 4 numbers are all 7’s...”

```

if ((v1 == 7) && (v2 == 7) && (v3 == 7) && (v4==7))

```

The following detects if there are any 3 variables having the value 7:

```

if (((v1 == 7) && (v2 == 7) && (v3 == 7)) ||
    ((v1 == 7) && (v2 == 7) && (v4 == 7)) ||
    ((v2 == 7) && (v3 == 7) && (v4 == 7)))

```

The following detects if there are any 3 variables having exactly the same values (any possible value will work):

```

if (((v1 == v2) && (v2 == v3)) ||
    ((v1 == v2) && (v2 == v4)) ||
    ((v2 == v3) && (v3 == v4)))

```

The following detects if there are any two variables having the same values (any possible value will work):

```

if ((v1 == v2) || (v1 == v3) ||
    (v1 == v4) || (v2 == v3) ||
    (v2 == v4) || (v3 == v4))

```

In the Pitching and Catching Baseball game, the game rules are:

- If the catcher uses his glove to catch the ball, the play wins 10 points.
- If the catcher does not catch the ball, the player loses 10 points.
- If the base ball moves out of the body area of the browser, the player get 5 points for bad pitching.

```

function play() {
  if ( bb.style.pixelLeft >= document.body.clientWidth-50) {
    bb.style.display="none"; score-=10; p1.innerText=score;}
  else {
    if ((bb.style.pixelLeft >= ct.style.pixelLeft + 50) &&
        (bb.style.pixelTop <= ct.style.pixelTop + 20) &&
        (bb.style.pixelTop >= ct.style.pixelTop)) {
      bb.style.display="none"; score+=10; p1.innerText=score;
    }
    else if ((bb.style.pixelTop >= document.body.clientHeight-20) ||
              (bb.style.pixelTop<=0)) {
      bb.style.display="none"; score+=5; p1.innerText=score;
      msg.innerText="Bad pitch, you got 5 points!";
    }
    else {
      bb.style.pixelLeft+=10;
      bb.style.pixelTop+=Math.tan(i);
      s1 = setTimeout("play()",50);
    }
  }
}

```

The following line simply means the baseball move over the catcher’s defending zone. In other words, the catcher does not catch the ball. The player loses 10 points.

```
if ( bb.style.pixelLeft >= document.body.clientWidth-50) {
    bb.style.display="none"; score-=10; p1.innerText=score;}

```

The following detects if the baseball moves through the glove area, as shown below.



```
if ((bb.style.pixelLeft >= ct.style.pixelLeft + 50) &&
    (bb.style.pixelLeft <= ct.style.pixelLeft + 77) &&
    (bb.style.pixelTop <= ct.style.pixelTop + 20) &&
    (bb.style.pixelTop >= ct.style.pixelTop)) {
    bb.style.display="none"; score+=10; p1.innerText=score;
}

```

If the above collision detection returns “True”, the player wins 10 points.

The following code determines if the base ball moves out of the body area of the browser. If so, the player gets 5 points and the screen displays “Bad pitch, you got 5 points!”

```
else if ((bb.style.pixelTop >= document.body.clientHeight-20) ||
    (bb.style.pixelTop<=0)) {
    bb.style.display="none"; score+=5; p1.innerText=score;
    msg.innerText="Bad pitch, you got 5 points!";
}

```

#### Review Questions

- Keeping score for a game player is a task \_\_\_\_\_.
  - of incrementing or decrementing the value of a variable
  - that requires the player to manually enter the score points to a textbox.
  - that requires the player to buy a USB scoreboard device and plugs it in to the computer.
  - All of the above
- You need to add 10 points to your scoreboard function of your game, which would you use?
  - score = score ++ 10;
  - score = score += 10;
  - score += 10;
  - score ++ 10;
- You need to deduct 1 point from your scoreboard function of your game, which would you use?
  - score -=;
  - score;
  - score -= score;
  - score -- score;
- Given the following code segment, which variable keeps a string literal?
 

```
var width = "20";
var height = 17;
var length = 129 + 11;
var area = height * 19;
```

- A. width
- B. height
- C. length
- D. area

5. Given the following code segment, which statement is correct?

```
if (f1.t1.value==f1.t2.value) { .. }
.....
<form name="f1">
.....
<input type="text" name="t1">
<input type="hidden" name="t2">
</form>
.....
```

- A. f1.t1.value represents the value that is entered to the t1 textbox.
- B. t2 is an invisible textbox inside the f1 HTML form.
- C. it checks if the value of t1 equals to the value of t2.
- D. All of the above

6. Given the following code segment, which is the value that will be displayed in the p1 object?

```
var score = 0;
score += 50;
score--;
score -= 5;
p1.innerText = score;
```

- A. 50
- B. 49
- C. 44
- D. 0

7. Given the following code segment, how many image file(s) will pop up?

```
<script>
function init() {
  for (i=1; i<=4; i++) {
    code1 = "<img src=b.gif>";
    area1.innerHTML = code1;
  }
}
</script>
<body onLoad=init()>
<div id="area1"></div>
</body>
```

- A. 1
- B. 2
- C. 3
- D. 4

8. Given the following code segment, how many image file(s) will pop up?

```
<script>
function init() {
  for (i=1; i<=4; i++) {
    code1 = "<img src=b.gif>";
  }
}
```

```

        area1.innerHTML += code1;
    }
</script>
<body onLoad=init()>
<div id="area1"></div>

```

- A. 1
- B. 2
- C. 3
- D. 4

9. Given the following code segment, how many image file(s) will pop up?

```

<script>
function init() {
var code1 = "";
for (i=1; i<=4; i++) {
    code1 += "<img src=b.gif>";
}
    area1.innerHTML = code1;
}
</script>
<body onLoad=init()>
<div id="area1"></div>

```

- A. 1
- B. 2
- C. 3
- D. 4

10. The following code literally means "\_\_\_".

```

if ((v1 == 7) && (v2 == 7) && (v3 == 7) && (v4==7))

```

- A. If 4 numbers are all 7's
- B. If there are any 3 numbers having 7 as values
- C. If there are any 2 numbers having 7 as values
- D. If there is any 1 number having 7 as value

## Game Programming

Lab #10

Score boards

### Preparation #1:

1. Create a new directory named **C:\games**.
2. Use Internet Explorer to go to <http://business.cypresscollege.edu/~pwu/cis261/download.htm> to download lab10.zip (a zipped) file. Extract the files to C:\games directory.

### Learning Activity #1: Guessing Number Game

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named **C:\games\lab10\_1.htm** with the following contents:

```
<html>

<style>
span {border:solid 2 black; width:30; height:30;
font-size:20px; text-align:center; color:white;
background-color:green;}
</style>

<script>
var score=0;
function init() {
  code1="";
  for (i=1; i<=64; i++) {
    if (i%8==0) {
      code1+="<span id=n"+i+">"+"i+"</span><br>";
    } else { code1+="<span id=n"+i+">"+"i+"</span>"; }
  }
  areal.innerHTML=code1;
  p1.innerText = 0;
}

function start() {
  var k=Math.floor(Math.random()*64);
  f1.t2.value=k;
  if (k==0 || k>64) {k=Math.floor(Math.random()*64);}

  for (i=1; i<=64; i++) {
    code1="n"+i+".style.backgroundColor='green';"
    eval(code1)
  }

  eval("n"+k+".style.backgroundColor='red';");

  sf = setTimeout("start()",50);
}

function stop() {
  clearTimeout(sf);
  if (f1.t1.value==f1.t2.value) { score+=1000; p1.innerText = score;}
  else { score -=1; p1.innerText = score;}
}
</script>
```

```

<body onLoad=init()>
<div id=areal></div>
<form name=f1>
Enter a number (1-64):
<input type=text name=t1 size=5>
<input type=hidden name=t2>
</form>
<p>Your Score: <b id=p1></b></p>
<button onClick="start()">Start</button>
<button onClick="stop()">Stop</button>
</body></html>

```

3. Test the program. A sample output looks:



## Learning Activity #2: Punching duckling

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab10\_2.htm with the following contents:

```

<html>
<style>
img {position: absolute; display:none}
</style>

<script>
var score=0;

function init() {
  for (i=1; i<=10; i++) {
    codes="<img id='d'+i+' ' src='duck.gif' style='left:10; top:10'>";
    areal.innerHTML += codes;
  }
  DuckMove()
}

function DuckMove() {

var w=Math.floor(document.body.clientWidth / 10);
for (i=1; i<=9; i++) {
  code1="if (d"+i+" style.pixelLeft >= w) ";
  code1+=" { d"+(i+1)+" style.display='inline'; ";
  code1+="d"+(i+1)+" style.pixelLeft += 5; }";
  eval(code1);
}

```

```

    if (d1.style.pixelLeft >= document.body.clientWidth-w)
    { d1.style.display = "none";}
    else { d1.style.display='inline';d1.style.pixelLeft += 5;}

for (j=2; j<=9; j++) {
    code2="if (d"+j+".style.pixelLeft >= document.body.clientWidth-w)";
    code2+=" { d"+j+".style.display = 'none';}";
    eval(code2);
}

    if (d10.style.pixelLeft >= document.body.clientWidth-w)
    { d10.style.display = 'none'; gameOver();}
    else { s1 = setTimeout("DuckMove()", 100); }

}

function gameOver() {
clearTimeout(s1);
alert("Game Over!");
}

function GloveMove() {
var e=event.keyCode;
    switch (e) {
        case 37:
            gv.style.pixelLeft-=2;
            break;
        case 39:
            gv.style.pixelLeft+=2;
            break;
        case 83:
            punch()
            break;
    }
}

function punch() {
gv.style.pixelLeft+=2;
gv.style.pixelTop-=2;

if (gv.style.pixelTop <=0)
{clearTimeout(s2);gv.style.pixelTop=200;gv.style.pixelLeft=10;}
else {s2=setTimeout("punch()", 20);}

    for (i=1; i<=10; i++) {
        code3="if ((gv.style.pixelLeft >= d"+i+".style.pixelLeft) &&";
        code3+=" (gv.style.pixelLeft <= (d"+i+".style.pixelLeft + 61)) && ";
        code3+=" (gv.style.pixelTop <= d"+i+".style.pixelTop)) ";
        code3+="{d"+i+".style.display='none';d"+i+".src='noduck.gif';";
        code3+="score+=10;p1.innerText=score}";
        eval(code3);
    }
}

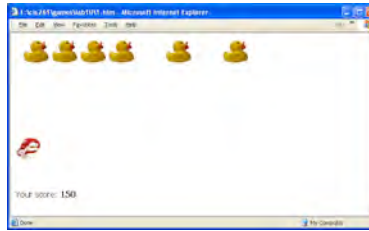
</script>

<body onLoad=init() onKeyDown=GloveMove() bgcolor="white">
<div id="areal" style="width:100%"></div>

<p style="position:absolute; top:300">Your score: <b id=p1></b></p>
</body></html>

```

3. Test the program. Press S to punch. Press → or ← to move to left or right. A sample output looks:



### Learning Activity #3: Shooting Crab Game

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab10\_3.htm with the following contents:

```
<html>
<style>
.bee {position:relative; z-index: 1}
</style>

<script>
var score=0;

function moveLeft() {
leftSide = areal.style.pixelLeft + 32;

if (leftSide >= 300) {
if (areal.style.pixelTop >= 200) {
clearTimeout(sf);
}
else {
areal.style.pixelTop += 20;
clearTimeout(sf); moveRight();
}
}
else {
areal.style.pixelLeft += 5;
sf = setTimeout("moveLeft()", 50);
}
}

function moveRight() {
if (areal.style.pixelLeft == 0) {
if (areal.style.pixelTop >= 200) {
clearTimeout(sr);
}
else {
areal.style.pixelTop += 20;
clearTimeout(sr); moveLeft();
}
}
else {
areal.style.pixelLeft -= 5;
sr = setTimeout("moveRight()", 50);
}
}
}
```

```

function shoot() {
var e=event.keyCode;
switch(e) {
case 37:
shooter.style.pixelLeft-=2; break;
case 39:
shooter.style.pixelLeft+=2; break;
case 83:
b1.style.pixelLeft = shooter.style.pixelLeft + 14;
b1.style.display = 'inline';
fire();
}
}

function fire() {
if (b1.style.pixelTop == 0) {
b1.style.display='none';
b1.style.top = 270;
clearTimeout(ss);

}
else {
b1.style.pixelTop -= 5;
if ((b1.style.pixelTop <= areal.style.pixelTop + 20) &&
(b1.style.pixelTop >=areal.style.pixelTop)) {
if ((b1.style.pixelLeft >= areal.style.pixelLeft) &&
(b1.style.pixelLeft <= areal.style.pixelLeft + 31))
{
c1.style.display='none';b1.style.display="none";
clearTimeout(ss); score += 10;
gd.innerText = score; newCrab() }
}
ss = setTimeout("fire()", 20);
}
}

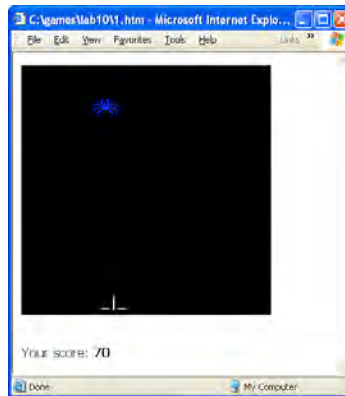
function newCrab() {
areal.style.pixelLeft = 0;
areal.style.pixelTop = 0;
c1.style.display='inline';
if (c1.style.backgroundColor=='yellow') {c1.style.backgroundColor='red';}
else if (c1.style.backgroundColor=='red') {c1.style.backgroundColor='blue';}
else if (c1.style.backgroundColor=='blue') {c1.style.backgroundColor='orange';}
else if (c1.style.backgroundColor=='orange') {c1.style.backgroundColor='white';}
else if (c1.style.backgroundColor=='white') {c1.style.backgroundColor='yellow';}
}
</script>

<body onLoad=moveLeft() onKeyDown=shoot(>
<div style="position:absolute; width:300; height:300; background-color: black">
<span id="areal" style="position:absolute;">
<img class='bee' id=c1 src='crab.gif' style="background-color:yellow" />
</span>
<span id="b1" style="position:absolute; top:270; display:none; color:white;
z-index:2">!</span>
<span id="shooter" style="position:absolute; top:275;
color:white"><b>_||_</b></span>

</div>
<p style="position:absolute;top: 350">Your score: <b id=gd></b></p>
</body>
</html>

```

3. Test the program. A sample output looks:



#### Learning Activity #4: Slot Machine

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab10\_4.htm with the following contents:

```
<html>

<style>
.box {position:relative; border:solid 5 black;
      font-size:72; font-family:arial; text-align:center;
      width:100; height:100}
</style>

<script>
var gd=0;

function init() {
codes="";
for (i=1; i<=4; i++) {
codes+="
```

```

        {gd += 30000; score.innerText = 30000;}

else if (((v1 == 7) && (v2 == 7) && (v3 == 7)) ||
        ((v1 == 7) && (v2 == 7) && (v4 == 7)) ||
        ((v2 == 7) && (v3 == 7) && (v4 == 7)))
    {gd += 3000; score.innerText = 3000;}

else if (((v1 == v2) && (v2 == v3)) ||
        ((v1 == v2) && (v2 == v4)) ||
        ((v2 == v3) && (v3 == v4))) { gd += 300; score.innerText = 300; }

else if ((v1 == v2) || (v1 == v3) ||
        (v1 == v4) || (v2 == v3) ||
        (v2 == v4) || (v3 == v4))
    {gd += 0; score.innerText = 0;}

else {gd -=30; score.innerText = "-30";}

credit.innerText = gd;
}
</script>

<body onLoad=init()>

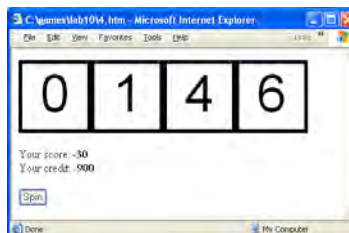
<div id="area1" style="position:absolute; background-color:white; width:400px;
height:100px">
<form name="f1">
<span id="area2" style="position:relative"></span>
</form>

<p>
Your score: <b id=score></b><br>
Your credit: <b id=credit></b><br>
</p>
<p><button onMouseDown="spin()" onMouseUp="check()">Spin</button>
</div>

</body>
</html>

```

3. Test the program. Press any key to spin. A sample output looks:



### Learning Activity #5: Pitching & Catching Baseball Game

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab10\_5.htm with the following contents:

```
<html>
```

```

<script>
var i=Math.floor(Math.random()*360);
var score=0;
function init() {
ct.style.left = document.body.clientWidth-150;
}

function check() {
var e=event.keyCode;
switch (e) {
case 83:
play(); break;
case 38:
ct.style.pixelTop -=5; break;
case 40:
ct.style.pixelTop +=5; break;
}
}

function play() {
if ( bb.style.pixelLeft >= document.body.clientWidth-50) {
bb.style.display="none"; score-=10; p1.innerText=score;}
else {
if ((bb.style.pixelLeft >= ct.style.pixelLeft + 50) &&
(bb.style.pixelTop <= ct.style.pixelTop + 20) &&
(bb.style.pixelTop >= ct.style.pixelTop)) {
bb.style.display="none"; score+=10; p1.innerText=score;
}
else if ((bb.style.pixelTop >= document.body.clientHeight-20) ||
(bb.style.pixelTop<=0)) {
bb.style.display="none"; score+=5; p1.innerText=score;
msg.innerText="Bad pitch, you got 5 points!";
}
else {
bb.style.pixelLeft+=10;
bb.style.pixelTop+=Math.tan(i);
s1 = setTimeout("play()",50);
}
}
}
}
</script>

<body onLoad=init(); onKeyDown=check()>



<p>Your score: <b id=p1></b><br>
<b id=msg><b></b></p>
</body>
</html>

```

3. Test the program. Press S to pitch. Use ↑ and ↓ keys to move the catcher. Use the browser's Refresh button to play the game again. Only when the ball hit the catcher's glove can you get  $10n$  points ( $n$  is an integer); otherwise you lose 10 points. If the ball moves out of the body area, you get 5 point for bad pitching. A sample output looks:



### Submittal

Upon completing all the learning activities,

1. Upload all files you created in this lab to your remote web server.
2. Log in to Blackboard, launch Assignment 10, and then scroll down to question 11.
3. Copy and paste the URLs to the textbox. For example,
  - [http://www.geocities.com/cis261/lab10\\_1.htm](http://www.geocities.com/cis261/lab10_1.htm)
  - [http://www.geocities.com/cis261/lab10\\_2.htm](http://www.geocities.com/cis261/lab10_2.htm)
  - [http://www.geocities.com/cis261/lab10\\_3.htm](http://www.geocities.com/cis261/lab10_3.htm)
  - [http://www.geocities.com/cis261/lab10\\_4.htm](http://www.geocities.com/cis261/lab10_4.htm)
  - [http://www.geocities.com/cis261/lab10\\_5.htm](http://www.geocities.com/cis261/lab10_5.htm)

No credit is given to broken link(s).

## Game Programming

### Lecture #11      Storyline and Texture

**Objective**      This lecture is aimed at walking students through some of the basic knowledge for developing complex games within a restricted coding environment. Along the way, this lecture provides real codes that will be useful to everyone who wants to develop basic skills for creating complex games.

**The Art of Storyboarding**      A storylines is the setting for the game, including such things as an introduction to the characters, their location, and the reason they do what they do. A storyline forms the spirit of the game.

In the arcade, you frequently see the dancing pad game. It is a game that uses is a flat electronic game controller for sending inputs to a dance platform (such as the one used in the arcade version of Dance Revolution).



Picture of Dance Revolution

To create a computer game as simulation of such game, you have the freedom to decide the storyline. In the following code example, the storylines are:

- Four arrows--left, right, up, and down--are selected randomly, one at a time.
- Change the color of the select arrow to red (from black).
- Detect the player's response using arrow keys on the keyboard.
- If the player presses the correct key, he/she wins 10 points.
- Additionally, add an animated dancing girl as 3D texture of this game.

The following codes randomly select a number from 1, 2, 3, and 4, which represent up, left, right, and down respectively. The **switch..case** statement, then determine which arrow's color must be changed to red. The **setTimeout()** method executes the **init()** function every 500 milliseconds.

```
function init() {
    clear();

    var i=Math.floor(Math.random()*4)+1;
    switch (i) {
        case 1:
            up.src="up_red.gif"; k=38;
            break;
        case 2:
            left.src="left_red.gif"; k=37;
            break;
        case 3:
            right.src="right_red.gif"; k=39;
            break;
        case 4:
            down.src="down_red.gif"; k=40;
            break;
    }
```

```

    }
    s1=setTimeout("init()", 500);
}

```

The **keyed()** function checks whether or not the player presses the correct arrow key on the keyboard by comparing the **keyCode** value with the value of a variable **k**. The **init()** function assigns a value to **k** every time when it is executed.

```

function keyed() {
if (event.keyCode==k) { score+=10; p1.innerText=score; }
}

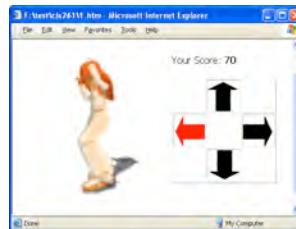
```

The **score** variable keeps score and let the **innerText** property insert the updates score to **p1**. The following line loads an animated gif file that delivers an effect of dancing girl.

```

```

A complete code of this game will screen a theme like:



As a programmer, it is easy for you to convert your thoughts into flows of game; however, the players may not be able to handle your game concepts if you fail to stick to the following unwritten rules:

- Start with a simple game theme. It is always best not to take risks with games that are too difficult and instead stick to a design that is simple and intuitive.
- Consider your target devices. Computers differ in CPU and memory sizes and color resolutions. A complicated game may not provide acceptable performance on all target computers.
- Develop a good storyline. Create a storyboard for every important screen of your game, so that you have a rough visual idea of your final product. And, develop a gripping back-story for supporting the gameplay. The storyline must be closely tied to the game view (top view, side view, isometric view), gameplay, game characters, levels, obstacles, game animations and so on.
- Focus on the right thing. Make sure that the back-story is not the focal point of the game. Too many background graphics or too complicate in texturing will take away the player's attention on the game. Textures should be created only if it is applicable to the game.

This lecture, from this point on, will focus on how to develop storylines and create game codes in according to the storylines.

#### Developing a storyline

Developing a storyline for a game usually consists of a few steps. First, decide what theme you want for your story. For example, in the Tetris game, you need to continuously drop shapes of bricks from the top to bottom. Each shape can be rotated horizontally or vertically, so you need to define all the possible shapes.

Your storyline for now will be:

- Display a square area with 16 cells in it.
- When the player press any key, make sure all the 16 cells' backgrounds are in white, and then change the background color of any predefined set of 4 cells to red.
- Be ready for the next theme.

Once the storyline is developed, you can begin writing code for this part of the entire storylines. Given the following code. It generates an output of 16 cells with numbers in each of them.

```
<html>
<style>
.cell {border:solid 1 black;
      width:20; height:20;
      background-Color: white}
</style>

<script>
function init() {
var code1="";
  for (i=1; i<=16; i++) {
    if (i%4==0) {
      code1 += "<span id=c"+i+" class='cell'>" + i + "</span><br>";
    } else {code1 += "<span id=c"+i+" class='cell'>" + i + "</span>";}
  }
  area1.innerHTML = code1;
}
</script>

<body onLoad=init()>
<div id="area1"></div>
</body>
</html>
```

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

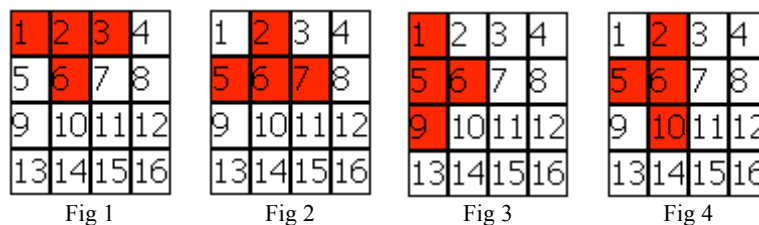
The following lines use the modulus operator (%), which returns the remainder of two numbers.

```
if (i%4==0) {
code1 += "<span id=c"+i+" class='cell'>" + i + "</span><br>";
} else {code1 += "<span id=c"+i+" class='cell'>" + i + "</span>";}
```

The above code uses the *for* loop to create 16 cells, but it only allows 4 cells each row. The modulus operator in this case helps to detect if the current value of *i* is a multiple of 4. If so, add `<br>` to `<span>..</span>` to break the line.

The modulus operator is also commonly used to take a randomly generated number and reduce that number to a random number on a smaller range, and it can quickly tell you if one number is a factor of another.

The above code only lays out the background area, which consists of 16 cells with numbers in them. Notice that these numbers are added only to help you figure out how you can design shapes (as those in the Tetris game). For example,



Logically speaking:

- To create the share of Fig 1, you need to change the background color of cell 1, 2, 3 and 6.
- To create the share of Fig 2, you need to change the background color of cell 2, 5, 6 and 7.
- To create the share of Fig 3, you need to change the background color of cell 1, 5, 6 and 9.
- To create the share of Fig 4, you need to change the background color of cell 2, 5, 6 and 10.

This part of code is included in **shapes()** function. Variables **n1**, **n2**, **n3**, and **n4** are used to represent the numbers of the 4 picked cells.

```
function shapes() {
  clear(); // call the clear() function
  var i=Math.floor(Math.random()*4)+1;
  switch (i) {
    case 1:
      n1=1; n2=2; n3=3; n4=6; break;
    case 2:
      n1=2; n2=5; n3=6; n4=7; break;
    case 3:
      n1=1; n2=5; n3=6; n4=9; break;
    case 4:
      n1=2; n2=5; n3=6; n4=10; break;
  }

  code2 = "c"+n1+".style.backgroundColor='red'";
  code2 += "c"+n2+".style.backgroundColor='red'";
  code2 += "c"+n3+".style.backgroundColor='red'";
  code2 += "c"+n4+".style.backgroundColor='red'";
  eval(code2);
}
```

If case 1 is selected (randomly by the computer), the following code blocks will become codes as shown in the dashed line area. Consequently, you see the shape of Fig 1 on screen.

```
code2 = "c"+n1+".style.backgroundColor='red'";
code2 += "c"+n2+".style.backgroundColor='red'";
code2 += "c"+n3+".style.backgroundColor='red'";
code2 += "c"+n4+".style.backgroundColor='red'";
eval(code2);

c1.style.backgroundColor='red';
c2.style.backgroundColor='red';
c3.style.backgroundColor='red';
c6.style.backgroundColor='red';
```

The **clear()** function is created to handle the “...make sure all the 16 cells’ backgrounds are in white...” part of the storyline.

```
function clear() {
  for (i=1; i<=16; i++) {
    code3 = "c"+i+".style.backgroundColor='white'";
    eval(code3);
  }
}
```

By adding the following functions, **shapes()** and **clear()**, you can randomly display a shape in four directions.

The next step is to decide how the theme will be played out. In a real Tetris game, you don’t see the numbers in cells, so modify the following line to prevent the numbers from being display (Compare with the above codes).

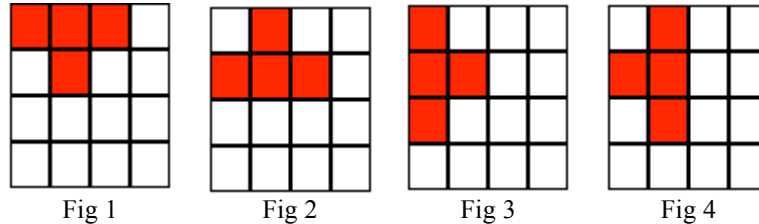
```
function init() {
  var code1="";
  for (i=1; i<=16; i++) {
```

```

    if (i%4==0) {
        code1 += "<span id=c"+i+" class='cell'></span><br>";
    } else {code1 += "<span id=c"+i+" class='cell'></span>";}
    }
    area1.innerHTML = code1;
}

```

The outputs now look:



Now that you have the theme set up, you can establish the protagonist and antagonist. The player is the protagonist in the Tetris game, while the computer surely is the antagonist. So, you need to decide the game rules for scoring, game stages (phases), and levels of difficulty, etc.. For example, you can modify the code of dancing pad game to let players choose the level of difficulty. Consider the following code,

```

function init() {
    clear();

    var i=Math.floor(Math.random()*4)+1;
    switch (i) {
        case 1:
            up.src="up_red.gif"; k=38;
            break;
        case 2:
            left.src="left_red.gif"; k=37;
            break;
        case 3:
            right.src="right_red.gif"; k=39;
            break;
        case 4:
            down.src="down_red.gif"; k=40;
            break;
    }
    s1=setTimeout("init()", 500);
}

```

Simply change to the re-execution time value from 500 to 1000, the player will have more time to decide what arrow key to press. When the value is lowered to 300, the player has less time to respond.

Seeing that you want to the player to select the level of difficulty--certainly one at a time, you can add a web form to collect the player's entry.

```

function init() {
    clear();
    var j=f1.t1.value;
    switch(j) {
        case 1:
            sec = 1000; break;
        case 2:
            sec = 500; break;
        case 3:

```

```

        sec = 300; break;
    }

    var i=Math.floor(Math.random()*4)+1;
    switch (i) {
    case 1:
        up.src="up_red.gif"; k=38;
        break;
    case 2:
        left.src="left_red.gif"; k=37;
        break;
    case 3:
        right.src="right_red.gif"; k=39;
        break;
    case 4:
        down.src="down_red.gif"; k=40;
        break;
    }
    s1=setTimeout("init()", j);
}
.....
.....
<form name="f1">
Level [1-3]: <input type="text" name="t1" size=3><br>
<button onclick="init()">Start</button>
</form>
.....

```

In the Tetris game, your storylines eventually must include moving the shapes downwards. This part of code, for example, can be completed by adding the following bold lines:

```

.....
function init() {
var code1="";
for (i=1; i<=64; i++) { // change the maximum to 64
.....
.....
code2 = "c"+n1+".style.backgroundColor='red'";
code2 += "c"+n2+".style.backgroundColor='red'";
code2 += "c"+n3+".style.backgroundColor='red'";
code2 += "c"+n4+".style.backgroundColor='red'";
eval(code2);

st1=setInterval("moveDown()", 1000);
}

function clear() {
for (i=1; i<=64; i++) { // change the maximum to 64
code3 = "c"+i+".style.backgroundColor='white'";
eval(code3);
}
}

function moveDown() {
clearInterval(st1);

if ((n1>=60) || (n2>=60) || (n1>=60) || (n2>=60)) {
clearTimeout(s1);
}
else {
clear();
n1+=4;
n2+=4;

```

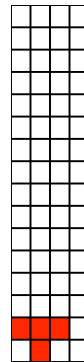
```

n3+=4;
n4+=4;

code4 = "c"+n1+".style.backgroundColor='red'";
code4 += "c"+n2+".style.backgroundColor='red'";
code4 += "c"+n3+".style.backgroundColor='red'";
code4 += "c"+n4+".style.backgroundColor='red'";
eval(code4);
s1 = setTimeout("moveDown()", 1000);
}
}

```

So a sample outcome now looks:

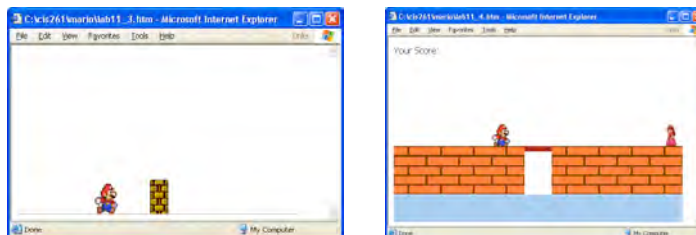


Your game will also include storylines that require computer to make human-like decisions. For example, in a Tic-Tac-Toe game, the player is the protagonist and the computer will serve the antagonist to play against the human player. The computer must be able to respond to the human player's movement with an intention to win the game. A later lecture will discuss about how you can program by using concepts of artificial intelligence to make your game code think like a human.

Developing storylines for multi-theme games

Before developing a multi-theme game, be sure to develop a linear sequence of themes with a smooth transition between any two of them.

In the **Super Mario** game, Mario needs to overcome two stages to rescue the princess. You will then need to develop two separated storylines--one for each stage..



You can create one single program with all the themes in it, or you can create separated programs on a one-for-each-theme basis. The decision should be made in accordance to the complexity of storyline. The more characters, antagonists, and levels of difficulty the game has, the more complicated it is.

In the Super Mario Theme 1 game, there is a **theme2()** function which tells the browser to change the web page to **lab11\_4.htm** (from **lab11\_3.htm**) when Mario jumps into the chimney.

```

function theme2() {
if (h1.style.pixelTop>=200) {

```

```

h1.style.display ='none';
clearTimeout(t1);
window.location='lab11_4.htm' } // change to the lab10_4.htm file
else {h1.style.pixelTop +=2;}
t1= setTimeout("theme2()", 20);
}

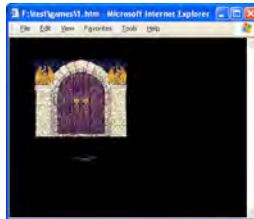
```

In the Bat Battle game, the storylines consist of two themes. In the first theme:

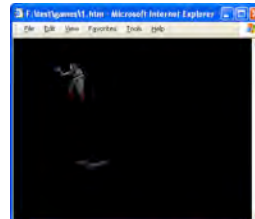
- A bat flies to a castle.
- When the bat approaches the gate, the gate opens by itself.

In the second theme:

- A bat flies toward a knight, who is watching the bat closely.
- When the timing is ready, the knight attacks that flying bat.



Theme 1



Theme 2

These two themes are placed in the same code with <body> and </body>, as shown below.

```
<body bgcolor="black" onLoad="init();flyBat()" />
```

```

// theme 1
<div id="theme1">




</div>

```

```

// theme 2

<div id="theme2" style="display:none">



</div>

```

```
</body>
```

Each theme is a panel by itself. A panel is a container that can include many objects. In the above code, <div> and </div> defines the two panels. Objects (such as image files) belonging to the first theme (the solid line area) are placed within:

```
<div id="theme1">.....</div>
```

Objects that belong to the second theme (the dashed line area) are placed within:

```
<div id="theme2" style="display:none">.....</div>
```

The second panel (the ID is “theme2”) is not displayed by default, because the **display** property has the value “**none**”. Thus, when the game starts, the player only sees the first theme.

The **flybat()** function controls the transition between theme 1 and 2, as shown below.

```
function flyBat() {
    if (bat.style.pixelWidth == 0) {
        theme1.style.display = "none";
        theme2.style.display = "inline";
        clearTimeout(s2); flyBat2();
    }
    else{
        bat.style.pixelWidth--;
        bat.style.pixelTop-=2;
        s2 = setTimeout("flyBat()", 50);
    }
}
```

The logic is simple--set the **display** property of theme 1 to “**none**”, and set the display property of theme 2 to “**inline**”. Consequently the first panel is no longer displayed, the second themes takes the place and is displayed on the screen.

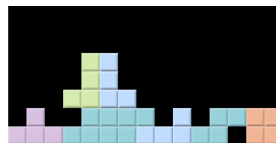
Noticeably, at the end of the following code block, the **flyBat2()** function is triggered, which launch the actions for theme 2.

```
if (bat.style.pixelWidth == 0) {
    theme1.style.display = "none";
    theme2.style.display = "inline";
    clearTimeout(s2); flyBat2();
}
```

The arrangement of this multi-theme is technically simple--whatever belongs to theme 1 is placed in the area for theme1, whatever belongs to theme 2 is placed in the area of theme 2. Between `<script>` and `</script>` tags, functions that will be used for theme 1 are tied together, while those for theme 2 are in another batch.

## Texturing and texture mapping

In graphics, the digital representation of the surface of an object is known as the **texture** of that graphics. In game programming, the effect of having a texture can be reached by applying a one-, two-, or three-dimensional image to a given object as surface. Such programming skill frequently requires the programmer to define a set of parameters that determine how visible surface are derived from the image. For example, by using some pre-designed 3D graphics with colorful texture as background, the Tetris game will look completely different (possibly more appealing to players).



In addition to two-dimensional qualities, such as color and brightness, a texture is also encoded with three-dimensional properties, such as how transparent and reflective the object is. Once a texture has been defined, it can be wrapped around any 3-dimensional object. This is called **texture mapping**.

Texture mapping is a technique that applies an image onto an object's surface as if the image were a decal or cellophane shrink-wrap. The image is created in texture space, with an (x, y, z) coordinate

system.

To add textures with DHTML, you can apply the so-called “**multimedia-style effects**” (Microsoft names them the “**visual filters**”) to standard HTML controls, such as text containers, images, and other non-window-specified objects.

By combining filters with JavaScript scripting, you can create visually engaging and interactive games. For example, the **alpha** filter can adjust the opacity of the content of the object. Its syntax is:

```
<HTMLTag STYLE=
  "filter:progid:DXImageTransform.Microsoft.Alpha(Str)"..>
```

where *str* is the string that specifies one or more properties exposed by the filter.

In the following code, the second image uses the alpha filter to fade out its right-bottom corner.

```



```

Compare the output of these two image file, the second image is **textured**.



When writing game codes, you can use the following syntax to dynamically control the object using visual filters:

```
ObjectID.style.filter =
  "progid:DXImageTransform.Microsoft.Alpha(str)";
```

For example, the following use the **light** filter to create a texture effect of a blue light shining on the content of the object.

```
<script>
window.onload=init;
function init() {
tp101.filters[0].addCone(0,0,1,250,499,0,0,255,20,180);
}
</script>


```

Surprisingly this is how this image looks now:



The **addCone()** method adds a cone light to the Light filter effect object to cast a directional light on the page. The syntax:

```
ObjectID.filters[n].addCone(iX1, iY1, iZ1, iX2, iY2, iRed, iGreen,  
iBlue, iStrength, iSpread);
```

where *n* is an integer that represents a filter in the order of an array. It is because an object can have more than one filter. The first filter has a key 0, the second 1, and so on.

All the parameters are:

<i>iX1</i>	Required. Integer that specifies the left coordinate of the light source.
<i>iY1</i>	Required. Integer that specifies the top coordinate of the light source.
<i>iZ1</i>	Required. Integer that specifies the z-axis level of the light source.
<i>iX2</i>	Required. Integer that specifies the left coordinate of the target light focus.
<i>iY2</i>	Required. Integer that specifies the top coordinate of the target light focus.
<i>iRed</i>	Required. Integer that specifies the red value. The value can range from 0 (lowest saturation) to 255 (highest saturation).
<i>iGreen</i>	Required. Integer that specifies the green value. The value can range from 0 (lowest saturation) to 255 (highest saturation).
<i>iBlue</i>	Required. Integer that specifies the blue value. The value can range from 0 (lowest saturation) to 255 (highest saturation).
<i>iStrength</i>	Required. Integer that specifies the intensity of the light filter. The value can range from 0 (lowest intensity) to 100 (highest intensity).
<i>iSpread</i>	Required. Integer that specifies the angle, or spread, between the vertical position of the light source and the surface of the object. The angle can range from 0 to 90 degrees. Smaller angle values produce a smaller cone of light; larger values produce an oblique oval or circle of light.

The cone light fades with distance from the target *x* and *y* position. The light displays a hard edge at the near edge of its focus and fades gradually as it reaches its distance threshold.

You can also integrate the item method with **addCode()** if you have more than one collections. The item method retrieves an object from the all collection or various other collections. Its syntax is:

```
ObjectID.item(vIndex [, iSubindex])
```

where ***vIndex*** is required. It should be an integer or string that specifies the object or collection to retrieve. If this parameter is an integer, it is the zero-based index of the object. If this parameter is a string, all objects with matching name or id properties are retrieved, and a collection is returned if more than one match is made.

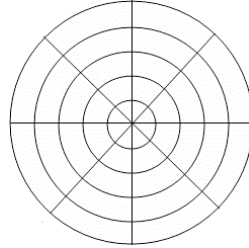
Also, ***iSubindex*** is optional. Integer that specifies the zero-based index of the object to retrieve

when a collection is returned.

The syntax to combine **item** and **addCode** methods is:

```
ObjectID.filters.item("DXImageTransform.Microsoft.Light").addCone(  
    iX1, iY1, iZ1, iX2, iY2, iRed, iGreen, iBlue, iStrength, iSpread);
```

In the Radar game, the **radar.gif** is a graphic with concentric circles and x, y axis. Its background color is white.



The code uses the **init()** function to create a radar-like texture and apply the texture to **radar.gif** file (its ID is **rd**). The **addCone** method generates the coned light beam.

```
function init() {  
    rd .filters.item('DXImageTransform.Microsoft.light').addCone(  
        121, 121, 0, Light_X, Light_Y, 0, 255, 0, 150, 10);  
    .....  
}  
.....  
.....  

```

In the movies, a radar screen normally uses blue as background color, so let the **addAmbient** method adds a blue background to the **rd** object.

```
function init() {  
    .....  
    rd .filters.item('DXImageTransform.Microsoft.light').addAmbient(  
        0, 0, 255, 80 )  
    .....  
}
```

The **addAmbient** method adds an ambient light to the Light filter. Ambient light is non-directional and distributed uniformly throughout space. Ambient light falling upon a surface approaches from all directions. The light is reflected from the object independent of surface location and orientation with equal intensity in all directions. Its syntax is:

```
addAmbient(iRed, iGreen, iBlue, iStrength);
```

where,

- **iRed** is an integer that specifies the red value. The value can range from 0 (lowest saturation) to 255 (highest saturation).
- **iGreen** is an integer that specifies the green value. The value can range from 0 (lowest saturation) to 255 (highest saturation).
- **iBlue** is an integer that specifies the blue value. The value can range from 0 (lowest saturation) to 255 (highest saturation).
- **iStrength** is an integer that specifies the intensity of the light filter. The value can range from 0 (lowest intensity) to 100 (highest intensity). The intensity specified pertains to the target coordinates.

On the radar screen, red points represent flying objects. To dynamically add an red point to the rd object, use the **addPoint** method.

```
function init() {
    .....
    rd.filters.item('DXImageTransform.Microsoft.Light').addPoint(
        PlaneLight_X, PlaneLight_Y, 3, 255, 0, 0, 100);
    .....
}
```

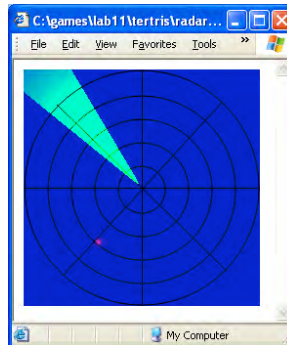
The **addPoint** method adds a light source to the Light filter. The light source originates at a single point and radiates in all directions.

```
addPoint(iX, iY, iZ, iRed, iGreen, iBlue, iStrength);
```

where,

- iX is an integer that specifies the left coordinate of the light source.
- iY is an integer that specifies the top coordinate of the light source.
- iZ is an integer that specifies the z-axis level of the light source.
- iRed is an integer that specifies the red value. The value can range from 0 (lowest saturation) to 255 (highest saturation).
- iGreen is an integer that specifies the green value. The value can range from 0 (lowest saturation) to 255 (highest saturation).
- iBlue is an integer that specifies the blue value. The value can range from 0 (lowest saturation) to 255 (highest saturation).
- iStrength is an integer that specifies the intensity of the light filter. The value can range from 0 (lowest intensity) to 100 (highest intensity). The intensity specified pertains to the target coordinates.

With the above codes, a dynamically created texture is applied to the rd object (radar.gif file), so it now looks:



Microsoft defined many visual filters and methods to support these filters. You can visit the Microsoft MSDN site for details about every filters. As of April, 2007, the URL is <http://msdn.microsoft.com/workshop/author/filter/reference/reference.asp?frame=true>

## Review Questions

1. Which provides a framework for computer game?
  - A. sprite
  - B. storyboard
  - C. storyline
  - D. genre
2. Given the following code segment, which assigns a file source "flower.gif" to it so the image will appear?

```
<img id="m1">
```

- A. `m1.style.src = "flower.gif"`
- B. `m1.src = "flower.gif"`
- C. `src(m1) = "flower.gif"`
- D. `src.m1 = "flower.gif"`

3. Which is the best way to let computer randomly pick a number from 1 to 4?

- A. `Math.floor(Math.random()*4);`
- B. `Math.floor(Math.random()*4 + 1);`
- C. `Math.floor(Math.random()*4) + 1;`
- D. `Math.floor(Math.random()*4) - 1;`

4. Given the following code segment, how will the "area1" object look?

```
var code1="";
for (i=1; i<=15; i++) {
  if (i%3==0) {
    code1 += "<span id=c"+i+" class='cell'></span><br>";
  }
  else {code1 += "<span id=c"+i+" class='cell'></span>";}
}
area1.innerHTML = code1;
```

- A. 3 rows 5 columns
- B. 5 rows 3 columns
- C. 3 rows 3 columns
- D. 5 rows 5 columns

5. Given the following code segment, which can change the background color of the "c1" object to red?

```
<span id="c1" style="width:100"></span>
```

- A. `c1.style.backgroundColor='red';`
- B. `c1.style.bgcolor='red';`
- C. `c1.backgroundColor='red';`
- D. `c1.style.bgcolor='red';`

6. Given the following code segment, if you wish to run the `init()` code at a slower but constant speed, you should \_\_\_\_.

```
var spd = 500;
s1=setTimeout("init()", spd);
```

- A. increase the value of `spd` variable
- B. decrease the value of `spd` variable
- C. let the value of `spd` variable increment by 10 repetitively
- D. let the value of `spd` variable decrement by 10 repetitively

7. Given the following code segment, which statement is correct?

```
st1=setInterval("init()", 1000);
```

- A. It sets a one-time pause for 1 second and then executes `init()`.
- B. It sets a repeating pause every 1 second before it executes `init()`.
- C. It sets an infinite pause every 1 second before it executes `init()`.
- D. It sets an endless pause every 1 second before it executes `init()`.

8. Which can hide a theme which is included in a pair of `<div id="theme1">` and `</div>` tags?
- A. `theme1.style.show = "false"`
  - B. `theme1.style.hide = "true"`
  - C. `theme1.style.display = "none"`
  - D. `theme1.style.visible = "none"`

9. Which Microsoft visual filter can fade out an image partially?
- A. beta
  - B. sigma
  - C. fade
  - D. alpha

10. Given the following code, which value specifies the intensity of the light filter?

```
addAmbient( 0, 120, 255, 80 )
```

- A. 0
- B. 120
- C. 255
- D. 80

## Game Programming

### Lab #11

### Storyline and Texture

#### Preparation #1:

1. Create a new directory named **C:\games**.
2. Use Internet Explorer to go to **<http://business.cypresscollege.edu/~pwu/cis261/download.htm>** to download lab11.zip (a zipped) file. Extract the files to C:\games directory.

#### Learning Activity #1: Dancing Pad Game

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named **C:\games\lab11\_1.htm** with the following contents:

```
<script>
var score = 0;
var k;

function init() {
clear();

var i=Math.floor(Math.random()*4)+1;
switch (i) {
case 1:
up.src="up_red.gif"; k=38;
break;
case 2:
left.src="left_red.gif"; k=37;
break;
case 3:
right.src="right_red.gif"; k=39;
break;
case 4:
down.src="down_red.gif"; k=40;
break;
}
sl=setTimeout("init()", 500);
}

function clear() {
up.src="up_black.gif";
left.src="left_black.gif";
right.src="right_black.gif";
down.src="down_black.gif";
}

function keyed() {
if (event.keyCode==k) { score+=10; p1.innerText=score; }
}
</script>

<body onLoad=init() onKeyDown=keyed() />

<table>
<tr><td>

</td>
```

```

<td>
<p>Your Score: <b id=p1></b></p>
<table border=1>
<tr><td></td>
<td></td>
<td></td></tr>

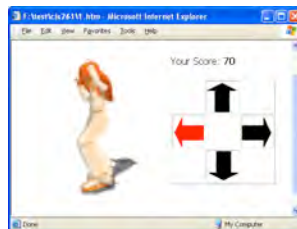
<tr><td></td>
<td></td>
<td></td></tr>

<tr><td></td>
<td></td>
<td></td></tr>

</td>
</table>

```

3. Test the program. Press the correct arrow keys on the keyboard as response to the red arrow on screen. A sample output looks:



## Learning Activity #2: Multi-theme game

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab11\_2.htm with the following contents:

```

<html>
<script>
// theme 1 functions
function init() {
if (bat.style.pixelTop <= 200) {
    castle.src="gate_open.gif";
    setInterval("castle.src='gate_opened.gif'",1600);
clearTimeout(s1);
}
else {
s1 = setTimeout("init()", 150);
}
}

function flyBat() {

if (bat.style.pixelWidth == 0) {
    theme1.style.display = "none";
    theme2.style.display = "inline";
    clearTimeout(s2); flyBat2();
}
}

```

```

else{
    bat.style.pixelWidth--;
    bat.style.pixelTop-=2;
    s2 = setTimeout("flyBat()", 50);
}
}

// theme 2 functions
function flyBat2() {

    if (bat2.style.pixelTop <= 170) {
        knight.src="knight2.gif";
        setInterval("knight.src='knight1.gif'", 4000);
    }
    if (bat2.style.pixelWidth == 0) {
        clearTimeout(s3);
    }
    else{
        bat2.style.pixelWidth--;
        bat2.style.pixelTop-=2;
        s3 = setTimeout("flyBat2()", 50);
    }
}

</script>

<body bgcolor="black" onLoad="init();flyBat()" />

// theme 1
<div id="themel">




</div>

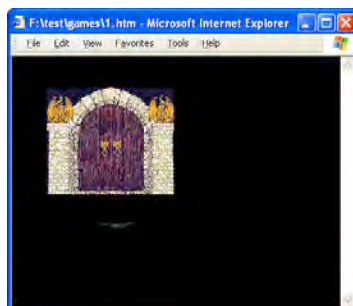
// theme 2
<div id="theme2" style="display:none">


</div>

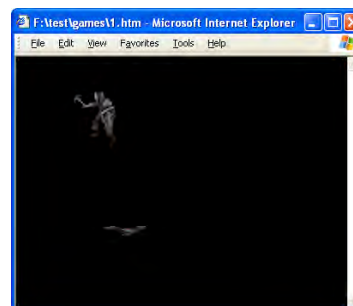
</body>
</html>

```

3. Test the program. A sample output looks:



Theme 1



Theme 2

### Learning Activity #3: Super Mario Theme 1 (this file and lab11\_4.htm are linked)

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named **C:\games\lab11\_3.htm** with the following contents:

```
<html>

<script>
var i=0;
var direction = "up";

function keyed() {
var e=event.keyCode;
switch (e) {
case 39:

    if ((h1.style.pixelLeft + 36 < b1.style.pixelLeft) ||
        (h1.style.pixelLeft > b1.style.pixelLeft + 18)) {
        h1.src="4.gif"; h1.style.pixelLeft+=2;
    }
    else {
        h1.style.pixelLeft = b1.style.pixelLeft - 36 ; h1.src="1.gif"; }
    break;

case 83:
    jump(); break;
}
}

function jump() {
h1.src="2.gif";
if (i==50) { direction = "down"; }

if (direction=="up") { h1.style.pixelTop -=2;
    if (i>=20) { h1.style.pixelLeft +=1;}
    i++; }
else {h1.style.pixelTop +=2; i--;}

if (i<0) {clearTimeout(s1);h1.style.pixelTop -=2;
    i=0; direction="up"; h1.src="1.gif"}
else {
s1 = setTimeout("jump()", 10);
}

if ((h1.style.pixelLeft + 36 >= b1.style.pixelLeft) &&
    (h1.style.pixelLeft + 36 >= b1.style.pixelLeft + 18) &&
    (h1.style.pixelTop + 46 >= b1.style.pixelTop)) {
    clearTimeout(s1);
    h1.src="1.gif";
    theme2();
}
}

function theme2() {
if (h1.style.pixelTop>=200) {
    h1.style.display ='none';
    clearTimeout(t1);
    window.location='lab11_4.htm'} // change to the lab10_4.htm file
```

```

else {h1.style.pixelTop +=2;}
t1= setTimeout("theme2()", 20);
}

function keyUp() {
if (h1.src=="4.gif") { h1.src="1.gif"; }
}

</script>

<body onKeyDown="keyed()" onKeyUp="keyUp()">


<hr size=1 width=400 style="position:absolute; top:243;">

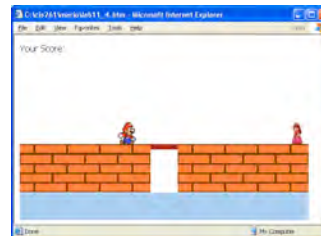
</body>
</html>

```

3. Test the program. Use → key to move toward the chimney, and then press S to jump into the chimney. A sample output looks:



Theme 1



Theme 2 (lab11\_4.htm)

#### Learning Activity #4: Super Mario Theme 2 (this file is the second theme of lab11\_3.htm)

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab11\_4.htm with the following contents:

```

<html>
<style>
.area {position:absolute}
</style>

<script>
var score=0;
var barNh1="false";

function init() {
for (i=1; i<=4; i++) {
code1 = "<img src='brick.gif'>";
area2.innerHTML += code1;
area3.innerHTML += code1;
}
moveBar();
}

function keyed() {
var e=event.keyCode;
switch (e) {

```

```

    case 39:
        walk();
        break;
    case 37:
        break;
    case 83:
        clearTimeout(s1);
        break;
}
}

function walk() {
w1 = area2.style.pixelLeft + 4*60;
w2 = area3.style.pixelLeft;
w3 = area3.style.pixelLeft + 4*60;

if (h1.style.pixelLeft > w2) { barNh1="false"; h1.style.pixelTop = 160;}
if ((barNh1=="false") && (h1.style.pixelLeft >= w1) && (h1.style.pixelLeft <= w2))
{
    End(); clearTimeout(s1); }
else if (h1.style.pixelLeft >= w3-60) {clearTimeout(s1);h1.src="1.gif"; }
else
{
    h1.src="4.gif";
    h1.style.pixelLeft +=2;
    s1 = setTimeout("walk()", 20);
}
}

function End() {
if (h1.style.pixelTop + 46 <= bar.style.pixelTop) {
    h1.src="1.gif"; barNh1 = "true";
}
else {
    if (h1.style.pixelTop == area2.style.pixelTop + 88) {
        h1.style.display = "none";
        clearTimeout(s2); score -=10;
        pl.innerText=score; }
    else {
        h1.src="1.gif";
        h1.style.pixelTop += 2;
        s2 = setTimeout("End()", 20); }
}
}

function moveBar() {
    if (barNh1 == "true") {
        h1.style.pixelTop = bar.style.pixelTop - 46;
    }
    if (bar.style.pixelTop <= 10) {
        bar.style.pixelTop = 290;}

bar.style.pixelTop -= 5;
s3 = setTimeout("moveBar()", 300);
}
</script>

<body onLoad=init() onKeyDown=keyed(>

<div id="area1">

<span id="area2" class="area" style="left:10; top:200"></span>
<span id="area3" class="area" style="left:300; top:200"></span>

```

```

<span id="area4" class="area" style="left:10; top:289; background-Color:#abcdef;
width:530; height:50"></span>


<hr id="bar" class="area" size=10 color="brown" style="left:250; top:290;
width:50;" />

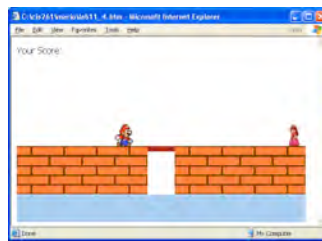
<p>Your Score: <b id=p1></b></p>

</div>

</body>
</html>

```

3. Test the program. Use → key to move toward the gap. Wait till the moving bar comes to an appropriate place. Use → key to step on the bar, and then jump to the second area to rescue the princess. A sample output looks:



### Learning Activity #5: Radar (dynamic texture and texture mapping)

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab11\_5.htm with the following contents:

```

<HTML>
<HEAD>

<SCRIPT language="JavaScript">
var Light_X = 20
var Light_Y = 20
var Light_Z = 40
var xInc = 10;
var yInc = 10;

var r = 100;
var deg = 0;
var deg1;

var rad;
var PointAngle;

var PlaneLight_X = 20
var PlaneLight_Y = 120
var PlaneLight_Z = 3
var conversion = (2 * Math.PI)/360

function movefilt()
{
    // Do some basic geometry to convert from Polar coordinates to Cartesian

```

```

Light_X = r + r * Math.cos(deg * conversion);
Light_Y = r + r * Math.sin(deg * conversion);

deg += 10;

if (deg == 360)
    deg = 0;

// Rotate the cone
rd .filters[0].moveLight(0, Light_X, Light_Y, Light_Z, 1);

// Figure out where the Plane is in relation to the cone
PointAngle = Math.atan((PlaneLight_Y - r)/(PlaneLight_X - r))/conversion;

// More basic geometry
if ((PlaneLight_X < r) && (PlaneLight_Y < r))
    PointAngle += 180;

if ((PlaneLight_X > r) && (PlaneLight_Y < r))
    PointAngle += 360

if ((PlaneLight_X < r) && (PlaneLight_Y > r))
    PointAngle += 180

// If the plane is in the cone, update the planes position
if ((deg - 10 <= PointAngle) && (PointAngle <= deg))
    rd .filters[0].moveLight(2, PlaneLight_X, PlaneLight_Y, PlaneLight_Z, 1);

//Do it all again in about 1/10th of a second
mytimeout=setTimeout('movefilt()', 100);

}

function movePlanes()
{
    // Increment the planes position
    PlaneLight_X++;
    PlaneLight_Y++;

    // Wrap the plane if it goes off the screen
    if (PlaneLight_Y > 200) PlaneLight_Y = 0;
    if (PlaneLight_X > 200) PlaneLight_X = 0;

    timeout2 = setTimeout('movePlanes()', 500);
}

function init() {
    rd .filters.item('DXImageTransform.Microsoft.light').addCone(121,121,0,Light_X,
    Light_Y, 0, 255, 0, 150, 10);
    rd .filters.item('DXImageTransform.Microsoft.light').addAmbient(0,0,255,80)
    rd .filters.item('DXImageTransform.Microsoft.light').addPoint(PlaneLight_X,
    PlaneLight_Y, 3, 255, 0, 0, 100);
    var x = 0;
    movefilt();
    movePlanes();
}
</SCRIPT>

</HEAD>

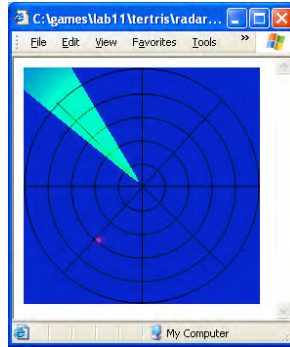
<BODY onload="init()" TOPMARGIN=10 LEFTMARGIN=10 bgcolor="white">

```

```


</BODY>
</HTML>
```

3. Test the program. A sample output looks:



### Submittal

Upon completing all the learning activities,

1. Upload all files you created in this lab to your remote web server.
2. Log in to Blackboard, launch Assignment 11, and then scroll down to question 11.
3. Copy and paste the URLs to the textbox. For example,
  - [http://www.geocities.com/cis261/lab11\\_1.htm](http://www.geocities.com/cis261/lab11_1.htm)
  - [http://www.geocities.com/cis261/lab11\\_2.htm](http://www.geocities.com/cis261/lab11_2.htm)
  - [http://www.geocities.com/cis261/lab11\\_3.htm](http://www.geocities.com/cis261/lab11_3.htm)
  - [http://www.geocities.com/cis261/lab11\\_4.htm](http://www.geocities.com/cis261/lab11_4.htm)
  - [http://www.geocities.com/cis261/lab11\\_5.htm](http://www.geocities.com/cis261/lab11_5.htm)

No credit is given to broken link(s).

## Game Programming

### Lecture #12 Applying artificial intelligence

**Concepts** Artificial intelligence (AI) is a relatively new subject in computer sciences, but it has been used in game programming for decades. AI refers to the programming techniques that make the computers emulate the human thought processes. The statement “applying artificial intelligence to a game” is simply the attempt to build a human-like decision making systems in a game. However, human thought is very complicated in terms of process. Even brain scientists do not know much about how human brains function, so the level of AI technology is still in its early infancy.

Many games, or even a section of storylines, require characters to be “smart” (or smart enough to respond to the changing condition. In the second theme of Castle game, when the bat approaches the knight, the bat (with its animal instincts) should know the knight is ready to hit it down. So the bat will keep a distance from the knight. On the other hand, the knight should be **smart** enough to move forward to a place where he can hit down the bat.

You can add the following codes for this scenario:

```
function hit() {
  if ( knight.style.pixelTop >=70) {
    clearTimeout(s2);
    setInterval("knight.src='knight1.gif'", 2000);
  }
  else {
    knight.style.pixelTop+=2;
    setInterval("bat2.style.display='none'", 2000);
    s2 = setTimeout("hit()", 20);
  }
}
```

By doing so, both characters--bat and knight--have more liveliness, because their actions and behaviors are much closer to animal and human.

**Information-based algorithms** Most traditional AI techniques use a variety of information-based algorithms to make decisions, just as people use a variety of previous experiences and mental rules to make a decision. The information-based AI algorithms were completely deterministic, which means that every decision could be traced back to a predictable flow of logic. This type of AI technique is the simplest way to apply artificial intelligence.

Consider the following example, the two fishes swim toward each other, and they will soon face to each other.

```
<html>
<style>
.fish {position:absolute;top:100;}
</style>

<script>
function init() {
f1.style.pixelLeft=10;
f2.style.pixelLeft=document.body.clientWidth-100;
move();
}

function move() {
  if (f1.style.pixelLeft+100 >= f2.style.pixelLeft) {
```

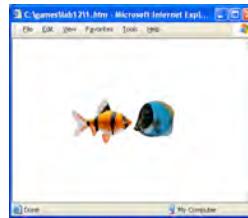
```

        clearTimeout(s1);}
    else {
        f1.style.pixelLeft+=2;
        f2.style.pixelLeft-=2;
        s1=setTimeout("move()", 20);
    }
}
</script>

<body onLoad=init()>


</body>
</html>

```



Assuming that fish1 is the predator of fish2, fish2's biological mechanism will force fish2 to avoid crashing into fish1. You can simulate such biological mechanism by adding the following code blocks:

```

function avoid() {
    if (f2.style.pixelTop >= document.body.clientHeight - 100) {
        clearTimeout(s2); }
    else {
        f2.style.pixelTop +=10;
        f2.style.pixelLeft-=2;
        f1.style.pixelLeft+=2;
        s2=setTimeout("avoid()", 10);
    }
}

```

In this example, the **if..then** statement formulates a very simple information-based AI, which helps the computer to make decisions. However, most games require a much complicated decision making system, especially when the computer must play the role of antagonist.

Obviously, the way you do your thinking is not never this simple and predictable. Using the deterministic approach to program games will never make the game think like a human. You can only try to apply as many strategies as you can to let the computer “understand” the condition, and then “find” the best solution to make decisions.

Developing a large-scale information-based AI for games

In a chess game, you fight against computer. As a human, your decisions can result from a combination of past experience, personal bias, or the current state of emotion in addition to the completely logical decision making process. So, there is technically no way to precisely predict how you will play the chess.

Luckily, there are strict rules a player must follow in order to play chess in an acceptable way. People can develop game strategies (which are basically the result of past experiences) and use them to make the best move against the antagonist.

As an AI-oriented game programmer, you need to convert such “human-developed game strategies” into computer codes, so the computer can also use these strategies against the human player. And, this is what “applying artificial intelligence to game programming” is all about.

Noticeably, human don't always make scientifically predictable decisions based on analyzing their surroundings and arriving at a logical conclusion. So, you probably will never be able to include every strategy in your games.

To develop a large-scale information-based AI for games starts with simply figuring out how to convert strategies into computer codes. For example, to make the Tic-Tac-Toe game "smart", first, create a framework for the Tic-Tac-Toe game by writing the following HTML codes:

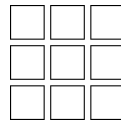
```
<html>


<br>


<br>


<br>
</html>
```

The output looks:



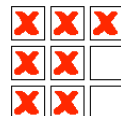
There are three image files used in this game:

- n.gif – a blank image
- x.gif – image of x
- o.gif – image of o

Inside each **<img>** tag the following part changes the file source (image file name) from **n.gif** to **x.gif**, and the **alt** property (which is the alternative description of the image file) from **n** to **x**.

```
onClick="this.src='x.gif';this.alt='x';check()"
```

When the player click on any cell, the image changes from **blank** to **x**. For example,



When the game starts, **x** is the player, while **o** is the antagonist (the computer). You need to add some codes to simulate human-like decision making mechanism, thus, whenever, the player makes a move, the computer can responds with an intention to prevent the user from winning the game.

To help developing the code with strategy against the player, try marking each cell with its IDs. For example,

c1	c2	c3
c4	c5	c6
c7	c8	c9

One commonly used strategy (as recommended by a sophisticated player) is:

- If c1 is marked x, while c2 and c4 are blank, check c3.
- If c3 is blank, mark c3 with o; otherwise mark c2 with o.

The code that performs the above human-like decision making is:

```
function check() {
  if (c1.alt=='x' && c2.alt=='n' && c4.alt=='n') {
    if (c3.alt=='n') { c3.src='o.gif'; }
    else { c2.src='o.gif'; }
  }
}
```

Consequently, when the player clicks c1 to mark **x** on the c1 cell, the above code immediately marks c3 with **o** to prevent the player from marking c1, c2, and c3 with **x**.

<b>x</b>		<b>o</b>

If the player marked c3 with **x** first, and then marks c1 with **x**, the code will be smart enough to mark c2 with **o**. In other words, the computer is now playing the game by making human-like decisions, so you have just applied **artificial intelligence** to the programming of this game.

<b>x</b>	<b>o</b>	<b>x</b>

Of course, this game cannot be so simple. You need to develop as many strategies to win as possible (or at least let the computer be smart enough to have a tie with the player). For example, a new strategy is “if c4 is marked **x**, and the player just marks c4 with **x**, then mark c7 with **o**.” Simply add the following bold lines, the computer can possess this “artificial intelligence”

```
function check() {
  if (c1.alt=='x' && c2.alt=='n' && c4.alt=='n') {
    if (c3.alt=='n') { c3.src='o.gif'; }
    else { c2.src='o.gif'; }
  }
  if (c1.alt=='x' && c2.alt=='n' && c4.alt=='x') {
    c7.src='o.gif';
}
}
```

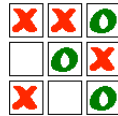
<b>x</b>		
<b>x</b>		
<b>o</b>		

Apply as many game strategies as you can to this Tic-Tac-Toe game, so the computer will be very smart and eventually have the intelligence to win over the player. For example, the following code is enough to reach a tie between the player and the computer if the player starts with clicking c1.

```

function check() {
  if (c1.alt=='x' && c2.alt=='n' && c4.alt=='n') {
    if (c3.alt=='n') { c3.src='o.gif'; }
    else { c2.src='o.gif'; }
  }
  if (c1.alt=='x' && c2.alt=='n' && c4.alt=='x') {
    c7.src='o.gif';
  }
  if (c1.alt=='x' && c2.alt=='x' && c5.alt=='n') {
    c5.src='o.gif';
  }
  if (c1.alt=='x' && c2.alt=='x' && c7.alt=='x') {
    c9.src='o.gif';
  }
}

```



Although the above strategies work with the demonstrated conditions, they may not be the best strategies. Be sure to develop your own game strategies to make this Tic-Tac-Toe game smart and possibly smarter.

When applying artificial intelligence to your games, one good advice **is to keep it as simple as possible**. If you know the answer, put the answer into the program. If you know how to compute the answer, put the algorithm for computing it into the program. For example, you can use the simplest **if..then** statement to re-write the above codes.

```

function check() {
  // player starts with c1
  if (c1.alt=='x' && c2.alt=='n' && c3.alt=='n' && c4.alt=='n') {
    c3.src='o.gif';
  }
  if (c1.alt=='x' && c2.alt=='x' && c3.alt=='n') { c3.src='o.gif'; }
  if (c1.alt=='x' && c2.alt=='n' && c3.alt=='x') { c2.src='o.gif'; }
  if (c1.alt=='x' && c4.alt=='x' && c7.alt=='n') { c7.src='o.gif'; }
  if (c1.alt=='x' && c4.alt=='n' && c7.alt=='x') { c4.src='o.gif'; }
  if (c1.alt=='x' && c5.alt=='x' && c9.alt=='n') { c9.src='o.gif'; }
  if (c1.alt=='x' && c2.alt=='x' && c5.alt=='n') { c5.src='o.gif'; }
  if (c1.alt=='x' && c2.alt=='x' && c5.alt=='n') { c5.src='o.gif'; }
  if (c1.alt=='x' && c4.alt=='x' && c8.alt=='x') { c5.src='o.gif'; }
  if (c1.alt=='x' && c2.alt=='x' && c6.alt=='x') { c7.src='o.gif'; }
  if (c1.alt=='x' && c2.alt=='x' && c8.alt=='x') { c7.src='o.gif'; }
  if (c1.alt=='x' && c2.alt=='x' && c9.alt=='x') { c7.src='o.gif'; }
  .....
  .....
}

```

In 1997, the famous supercomputer--IBM's Deep Blue, defeated the world great chess master Garry Kasparov. It marked the first time that a computer had defeated a World Champion in a match of several games. However, in reality, Deep Blue is just a computer that had been

programmed with millions of chess game strategies (developed by human in the past, of course).



IBM Deep Blue vs. Garry Kasparov

It is the computation power of the machine and the “artificial intelligence” the computer was programmed to have won the game. But, be sure to understand one important fact--Deep Blue do not think, it only quickly run through all the decision-making codes (just like the ones above) to search for the best-fit rules and then make decision based on the rules.

Create games  
that learns

You can create games that detect and record the player’s skill level, playing style, or thinking model, etc. You can also create games that can modify the game environment in according to the collected data about the player. There are many ways to reach this goal.

You can, for example, use a server-side scripting language (such as PHP, Perl, ASP.Net, etc.) to create a file (or set a cookie file) in the client computer that keeps data for the player. In the Ping Pong game, the Write\_level() function forces the client computer to create a new file (if not existing) named MyLevel.txt and immediately write a line “level=2” to it.

```
function play() {  
  if (score > 50) { speed = 10; level=2; Write_level();}  
  else {speed = 20; level=1}  
  .....  
  .....  
}  
  
function Write_level(sText){  
  var MYFILE = new ActiveXObject("Scripting.FileSystemObject");  
  var myLevel = MYFILE.CreateTextFile("C:\\MyLevel.txt", true);  
  myLevel.WriteLine("level=2");  
  myLevel.Close();  
}
```

This function uses ActiveXObject, which is an advanced topic for Windows-based web programming. However, you can learn the concepts as how to insert a file to a computer machine. Additionally, if the MyLevel.txt file already exists, and you need to update the content, you should use the following format:

```
var myLevel = MYFILE.OpenTextFile("C:\\MyLevel.txt", 8, true,0);
```

where, 8=append, true=create if not exist, and 0 = ASCII.

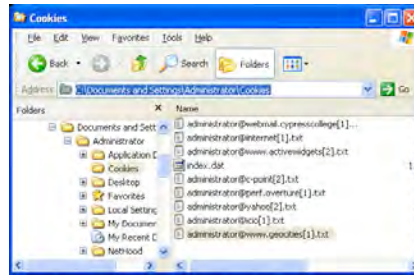
To set a cookie, for example, you can use the following code:

```
function setCookie() {  
  document.cookie = "level=2; expires=Fri, 13 Jul 2009 05:28:21 UTC;  
  path=/";  
}
```

The cookie code only works after you upload the file to a Web server. In other words, only when the player access the codes from a remote web server will the cookie file be inserted to the client machine. The path to file the cookie is usually (Windows XP):

```
C:\Documents and Settings\%UserName%\Cookies
```

Where %UserName% is the name the player uses to log in to the Windows XP machine. For example,



A sample content of the cookie looks:

```
level
2
www.geocities.com/
1088
3155626112
30016378
843931136
29854209
*
```

If your game uses client-side scripting language (such as JavaScript), you can use variables to store data temporarily, or create hidden fields of web forms to store data temporarily. Some client-side scripting languages (such as JavaScript) allows you to set a cookie in a client machine, too. For example, in the Ping Pong game, several variables are used to store the player's data temporarily. The speed of ping pong ball is determined by the score of the player. If the score is greater than 50, the speed increases and the level is upgraded to 2.

```
.....
function play() {
if (score > 50) { speed = 10; level=2; Write_level(); }
else {speed = 20; level=1}

if (b1.style.pixelLeft > bd.style.pixelWidth-5) {
clearTimeout(s1); p1.innerText="You lose!"; }
else {s1=setTimeout("play()", speed); }
.....
}
```

#### Review Questions

- Which statement best describes what "applying artificial intelligence to a game" is?
  - It describes a game that has intelligence.
  - It makes the computer functions like a real human.
  - It attempts to build a robot game.
  - It is simply the attempt to build a human-like decision making systems in a game.
- Which statement is correct about the term "information-based AI algorithms"?
  - It is completely referencing.
  - It is completely deterministic.
  - It is completely self-controlled.
  - It is completely empirical.
- Techniques of artificial intelligence can apply to a game that \_\_\_\_\_.
  - mimics an animal's self protection mechanism

- B. plays chess game against a real human player
- C. determine the level the player should play
- D. All of the above

4. Which statement is correct?

- A. Human logics are complicated, so you should try to make your computer code as complicated as possible.
- B. Although Human logics are complicated; you should try to make your computer code as simple as possible.
- C. Human logics are complicated, you should always use "switch..case" statement to apply AI to game codes.
- D. All of the above

5. Given the following code block of a Tic-Tac-Toe game, which statement is correct?

Let ci.alt=x represent X, let ci.alt='n' represent blank (where i is an integer between 1 and 9).

```
if (c1.alt=='x' && c2.alt=='n' && c3.alt=='n' && c4.alt=='n')
{ c3.src='o.gif'; }
```

- A. If the player places x on c1, while c2, c3, and c4 are blank, the computer placed o on c3.
- B. If the player places x on c1, while c2, c3, and c4 are NOT blank, the computer placed o on c3.
- C. If the player places X on c1, while c2, c3, and c4 are all marked with the letter 'n', the computer placed o on c3.
- D. If the player places X on c1, while c2, c3, and c4 are all marked with the letter 'n', the computer marks the letter 'o' on c3.

6. Given the following code block of a Tic-Tac-Toe game, which statement is correct?

Let ci.alt=x represent X, let ci.alt='n' represent blank (where i is an integer between 1 and 9).

```
if (c5.alt=='x' && c7.alt=='x' && c3.alt=='n') { c3.src='o.gif'; }
```

- A. If the player marked x on c5, and later marks x on c7, the computer must place o on c3 even when c3 is not blank.
- B. If the player marked x on c5, and later marks x on c7, the computer must place o on c3 only when c3 is blank.
- C. If the player marked o on c5, and later marks o on c7, the computer must place x on c3 even when c3 is not blank.
- D. If the player marked o on c5, and later marks o on c7, the computer must place x on c3 only when c3 is blank.

7. Given the following code block of a Tic-Tac-Toe game, which statement is correct?

Let ci.alt=x represent X, let ci.alt='n' represent blank (where i is an integer between 1 and 9).

```
if (c3.alt=='x' && c5.alt=='x' && c7.alt=='n') { c7.src='o.gif'; }
```

- A. When c3 and c5 are marked x, the computer must check whether or not c7 is marked o to decide what the next move should be.
- B. When c3 and c5 are marked x, the computer must check whether or not c7 is marked x to decide what the next move should be.
- C. When c3 and c5 are marked x, the computer must check whether or not c7 is blank to decide what the next move should be.
- D. When c3 and c5 are marked n, the computer must check whether or not c7 is marked n to decide the next move.

8. Given the following code segment, which statement is correct?

```
function cc(sText){
```

```

var f = new ActiveXObject("Scripting.FileSystemObject");
var s = f.CreateTextFile("C:\\s.txt", true);
s.WriteLine("Write_file");
s.Close();
}

```

- A. It creates a new file named s.txt in the client machine.
- B. It writes the message "write\_file" to the newly created file.
- C. It uses ActiveXObject.
- D. All of the above.

9. Given the following code segment, which statement is correct?

```

var obj = f.OpenTextFile("C:\\f1.txt", 8, true,0);

```

- A. 8=append
- B. true=create if not exist
- C. 0 = ASCII
- D. All of the above

10. Which would insert a cookie to the client machine?

- A. document.cookie();
- B. window.cookie()
- C. setCookie();
- D. insertCookie();

## Game Programming

### Lab #12

### Applying artificial intelligence

#### Preparation #1:

1. Create a new directory named **C:\games**.
2. Use Internet Explorer to go to **<http://business.cypresscollege.edu/~pwu/cis261/download.htm>** to download lab12.zip (a zipped) file. Extract the files to C:\games directory.

#### Learning Activity #1:

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named **C:\games\lab12\_1.htm** with the following contents:

```
<html>
<script>

function init() {

    if (bat2.style.pixelTop <= 170) {
        knight.src="knight2.gif";
    }
    if (bat2.style.pixelWidth == 20) {
        clearTimeout(s1);
        setInterval("hit()",1000);
    }
    else{
        bat2.style.pixelWidth--;
        bat2.style.pixelTop-=2;
        s1 = setTimeout("init()", 50);
    }
}

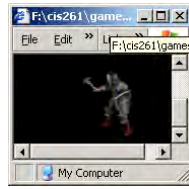
function hit() {
    if ( knight.style.pixelTop >=70) {
        clearTimeout(s2);
        setInterval("knight.src='knight1.gif'", 2000);
    }
    else {
        knight.style.pixelTop+=2;
        setInterval("bat2.style.display='none'", 2000);
        s2 = setTimeout("hit()", 20);
    }
}
}
</script>

<body bgcolor="black" onLoad="init()" />
<div id="theme2">


</div>

</body>
</html>
```

3. Test the program. A sample output looks:



## Learning Activity #2: Fish

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named **C:\games\lab12\_2.htm** with the following contents:

```
<html>
<style>
.fish {position:absolute;}
</style>

<script>
function init() {
  f1.style.pixelLeft=10;
  f2.style.pixelLeft=document.body.clientWidth-100;
  move();
}

function move() {
  if (f1.style.pixelLeft+100 >= f2.style.pixelLeft-100) {
    clearTimeout(s1); avoid();}
  else {
    f1.style.pixelLeft+=2;
    f2.style.pixelLeft-=2;
    s1=setTimeout("move()", 20);
  }
}

function avoid() {
  if (f2.style.pixelTop >= document.body.clientHeight - 100) {
    clearTimeout(s2); }
  else {
    f2.style.pixelTop +=10;
    f2.style.pixelLeft-=2;
    f1.style.pixelLeft+=2;
    s2=setTimeout("avoid()",10);
  }
}
</script>

<body onLoad=init()>


</body>
</html>
```

3. Test the program. A sample output looks:



### Learning Activity #3: Tic-Tac-Toe

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game. **Please feel free to add as many strategies as possible.**

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab12\_3.htm with the following contents:

```
<html>

<style>
.cell {border:solid 1 black; width:30px; height:30px}
</style>

<script>
function check() {

// player starts with c1
if (c1.alt=='x' && c2.alt=='n' && c3.alt=='n' && c4.alt=='n') { c3.src='o.gif'; }
if (c1.alt=='x' && c2.alt=='x' && c3.alt=='n') { c3.src='o.gif'; }
if (c1.alt=='x' && c2.alt=='n' && c3.alt=='x') { c2.src='o.gif'; }
if (c1.alt=='x' && c4.alt=='x' && c7.alt=='n') { c7.src='o.gif'; }
if (c1.alt=='x' && c4.alt=='n' && c7.alt=='x') { c4.src='o.gif'; }
if (c1.alt=='x' && c5.alt=='x' && c9.alt=='n') { c9.src='o.gif'; }
if (c1.alt=='x' && c2.alt=='x' && c5.alt=='n') { c5.src='o.gif'; }
if (c1.alt=='x' && c2.alt=='x' && c5.alt=='n') { c5.src='o.gif'; }
if (c1.alt=='x' && c4.alt=='x' && c8.alt=='x') { c5.src='o.gif'; }
if (c1.alt=='x' && c2.alt=='x' && c6.alt=='x') { c7.src='o.gif'; }
if (c1.alt=='x' && c2.alt=='x' && c8.alt=='x') { c7.src='o.gif'; }
if (c1.alt=='x' && c2.alt=='x' && c9.alt=='x') { c7.src='o.gif'; }

if (c1.alt=='x' && c6.alt=='x' && c5.alt=='n') { c5.src='o.gif'; }
if (c1.alt=='x' && c6.alt=='x' && c9.alt=='x') { c7.src='o.gif'; }
if (c1.alt=='x' && c9.alt=='x' && c5.alt=='n') { c5.src='o.gif'; }

if (c1.alt=='x' && c8.alt=='x' && c5.alt=='n') { c5.src='o.gif'; }
if (c1.alt=='x' && c5.alt=='x' && c8.alt=='x') { c6.src='o.gif'; }
if (c1.alt=='x' && c5.alt=='x' && c6.alt=='x') { c4.src='o.gif'; }

// player starts with c2
if (c1.alt=='n' && c2.alt=='x' && c3.alt=='n' && c4.alt=='n') { c1.src='o.gif'; }
if (c2.alt=='x' && c5.alt=='x' && c8.alt=='n') { c8.src='o.gif'; }
if (c5.alt=='x' && c7.alt=='x' && c3.alt=='n') { c3.src='o.gif'; }
if (c2.alt=='x' && c4.alt=='x' && c5.alt=='n') { c5.src='o.gif'; }
if (c2.alt=='x' && c4.alt=='x' && c5.alt=='x') { c6.src='o.gif'; }
if (c2.alt=='x' && c3.alt=='x' && c9.alt=='n') { c9.src='o.gif'; }
if (c2.alt=='x' && c5.alt=='x' && c6.alt=='x') { c4.src='o.gif'; }
if (c2.alt=='x' && c5.alt=='x' && c9.alt=='x') { c7.src='o.gif'; }

// Player starts with c3
if (c1.alt=='n' && c2.alt=='n' && c3.alt=='x' && c7.alt=='n') { c1.src='o.gif'; }
if (c3.alt=='x' && c6.alt=='x' && c9.alt=='n') { c9.src='o.gif'; }
if (c3.alt=='x' && c5.alt=='x' && c7.alt=='n') { c7.src='o.gif'; }
if (c3.alt=='x' && c4.alt=='x' && c5.alt=='x') { c6.src='o.gif'; }
```

```
// Player starts with c4
if (c1.alt=='n' && c4.alt=='x' && c7.alt=='n' && c5.alt=='n') { c5.src='o.gif'; }
if (c4.alt=='x' && c5.alt=='x' && c8.alt=='x' && c2.alt=='n') { c2.src='o.gif'; }
if (c4.alt=='x' && c7.alt=='x' && c9.alt=='x' && c8.alt=='n') { c8.src='o.gif'; }
if (c1.alt=='n' && c2.alt=='x' && c4.alt=='x') { c1.src='o.gif'; }
if (c1.alt=='n' && c4.alt=='x' && c7.alt=='x') { c1.src='o.gif'; }
if (c4.alt=='x' && c8.alt=='x' && c9.alt=='n') { c9.src='o.gif'; }
if (c4.alt=='x' && c3.alt=='x' && c9.alt=='x') { c6.src='o.gif'; }
if (c4.alt=='x' && c9.alt=='x' && c7.alt=='n') { c7.src='o.gif'; }
if (c4.alt=='x' && c6.alt=='x' && c3.alt=='n') { c3.src='o.gif'; }

}
</script>

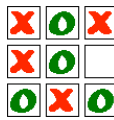


<br>


<br>


<br>
</html>
```

3. Test the program. A sample output looks:



#### Learning Activity #4: Ping Pong Game

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab12\_4.htm with the following contents:

```
<html>
<script>
var score=0;
var speed;
var level;
var left_right = "right";
var up_down = "down";

function init() {
bl.style.pixelTop = Math.floor(Math.random()*235);
bl.style.display = "inline";
p1.innerText = "";
```

```

play();
}

function move() {
ply.style.pixelTop = event.clientY;
}

function play() {
if (score > 50) { speed = 10; level=2; Write_level();}
else {speed = 20; level=1}

if (b1.style.pixelLeft > bd.style.pixelWidth-5) {
clearTimeout(s1); p1.innerText="You lose!"; }
else {s1=setTimeout("play()", speed);}

if ((b1.style.pixelLeft+15 >= ply.style.pixelLeft) &&
(b1.style.pixelTop > ply.style.pixelTop) &&
(b1.style.pixelTop+15 < ply.style.pixelTop+50))
{ left_right = "left"; score+=10; p1.innerText=score; p2.innerText=level;}

if (b1.style.pixelLeft <= 10) { left_right = "right"; }
switch (left_right) {
case "right":
b1.style.pixelLeft++; break;
case "left":
b1.style.pixelLeft--; break;
}

if (b1.style.pixelTop <=10) { up_down="down"; }
if (b1.style.pixelTop >=250) { up_down="up"; }
switch (up_down) {
case "up":
b1.style.pixelTop--; break;
case "down":
b1.style.pixelTop++; break;
}
}

function Write_level(sText){
var MYFILE = new ActiveXObject("Scripting.FileSystemObject");
var myLevel = MYFILE.CreateTextFile("C:\\MyLevel.txt", true);
myLevel.WriteLine("level=2");
myLevel.Close();
}
</script>

<body onMouseMove="move()">

<!-- b1-ball, bd-board, ply-player -->
<img id="b1" SRC="ball.gif" style="position:absolute; Left:10; Width:15;
Height:15;z-index=3; display: none">

<div id="bd" style="position: absolute; Top:10; Left:10; Width:350; Height:250;
background-Color:green; z-index=2;"></div>

<span id="ply" style="position: absolute; Top:10; Left:340; Width:15; Height:50;
background-Color: blue; z-index:3"></span>

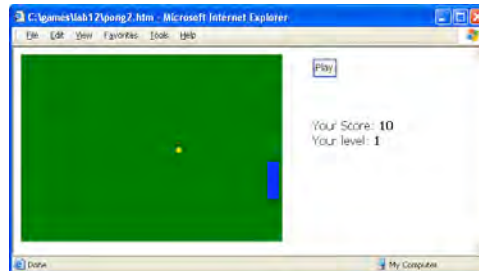
<button TYPE="button" onClick="init()" style="position: absolute; Top:15;
Left:400;z-index:3">Play</button>

<p style="position: absolute; Top:95; Left:400;z-index:3">Your Score: <b
id=p1></b></p>

```

```
<p style="position: absolute; Top:115; Left:400;z-index:3">Your level: <b
id=p2></b></p>
</body>
</html>
```

3. Test the program. Press the Play button to begin. Move mouse cursor up or down to move the blue bar. When your score is bigger than 50, the computer automatically upgrades your level to 2. A sample output looks:



### Learning Activity #5: Tetris

Note: This game is meant to be simple and easy for the sake of demonstrating programming concepts. Please do not hesitate to enhance the appearance or functions of this game.

1. Change to the C:\games directory.
2. Use Notepad to create a new file named C:\games\lab12\_5.htm with the following contents:

```
<HTML>
<style>
.MB { border: solid 1 white; background-color: red; height: 22px; width: 22px }
.SB { border: solid 1 white; background-color: teal; height: 22px; width: 22px
}
.BK { border: solid 1 #abcdef; height: 22px; width: 22px }
.GT { border: solid 1 black; }
</style>

<script>

var BX=new Array(4);
var BY=new Array(4);
var PX=new Array(4);
var PY=new Array(4);
var mTimer
var firstView

function init() {
var code1="<table cellpadding=0 cellspacing=0 class=gt>";
code1 += "<tbody id='GameBar'>";
for (i=1; i<=160; i++) {
if (i%10==1) { code1 += "<tr><td class=BK>&nbsp;</td>"; }
else if (i%10==0) { code1 += "<td class=BK>&nbsp;</td></tr>"; }
else { code1 += "<td class=BK>&nbsp;</td>"; }
}
code1 += "</tbody></table>";
areal.innerHTML += code1;

var code2="<table cellpadding=0 cellspacing=0 class=gt>";
code2 += "<tbody id='previewBar'>";
for (j=1; j<=16; j++) {
if (j%4==1) { code2 += "<tr><td class=BK>&nbsp;</td>"; }
else if (j%4==0) { code2 += "<td class=BK>&nbsp;</td></tr>"; }
else { code2 += "<td class=BK>&nbsp;</td>"; }
}
```

```

    }
    code2 += "</tbody></table>";
    area2.innerHTML += code2;
}

function beginGame()
{
    gameState=0;
    speed=1;
    outTime=1100-speed*100;
    if(gameState!=0) return;
    firstView=true;
    for(j=0;j<16;j++)
        for(i=0;i<10;i++)
            setClass(i,j,"BK");
    randBar();
    gameState=1;
    Play.disabled=true;
    window.clearInterval(mTimer);
    mTimer=window.setInterval("moveBar()",outTime);
}

function keyControl()
{
    if(gameState!=1) return;
    switch(event.keyCode){
        case 37: //left
            for(i=0;i<4;i++) if(BX[i]==0) return;
            for(i=0;i<4;i++) if(getClass(BX[i]-1,BY[i])=="SB") return;
            for(i=0;i<4;i++) setClass(BX[i],BY[i],"BK");
            for(i=0;i<4;i++) BX[i]=BX[i]-1;
            for(i=0;i<4;i++) setClass(BX[i],BY[i],"MB");
            break;
        case 38: //up
            var preMBarX=new Array(4);
            var preMBarY=new Array(4);
            var cx=Math.round((BX[0]+BX[1]+BX[2]+BX[3])/4);
            var cy=Math.round((BY[0]+BY[1]+BY[2]+BY[3])/4);
            for(i=0;i<4;i++){
                preMBarX[i]=Math.round(cx-cy+BY[i]);
                preMBarY[i]=Math.round(cx+cy-BX[i]);
                if(preMBarX[i]<0 || preMBarX[i]>9 || preMBarY[i]<0 ||
                    preMBarY[i]>15)
                    return;
                if(getClass(preMBarX[i],preMBarY[i])=="SB") return;
            }
            for(i=0;i<4;i++) setClass(BX[i],BY[i],"BK");
            for(i=0;i<4;i++){
                BX[i]=preMBarX[i];
                BY[i]=preMBarY[i];
            }
            for(i=0;i<4;i++) setClass(BX[i],BY[i],"MB");
            break;
        case 39: //right
            for(i=0;i<4;i++) if(BX[i]==9) return;
            for(i=0;i<4;i++) if(getClass(BX[i]+1,BY[i])=="SB") return;
            for(i=0;i<4;i++) setClass(BX[i],BY[i],"BK");
            for(i=0;i<4;i++) BX[i]=BX[i]+1;
            for(i=0;i<4;i++) setClass(BX[i],BY[i],"MB");
            break;
        case 40: //down
            moveBar();
            break;
    }
}

```

```

    }
}

function delLine()
{
    for(i=0;i<4;i++) setClass(BX[i],BY[i],"SB");
    for(j=0;j<16;j++){
        dLine=true;
        for(i=0;i<9;i++){
            if(getClass(i,j)!="SB"){
                dLine=false;
                break;
            }
        }
        if(dLine){
            for(k=j;k>0;k--){
                for(l=0;l<10;l++){
                    setClass(l,k,getClass(l,k-1));
                }
                for(l=0;l<10;l++) setClass(l,0,"BK");
            }
        }
        randBar();
        window.clearInterval(mTimer);
        mTimer=window.setInterval("moveBar()",outTime);
    }

function getClass(x,y){return GameBar.children[y].children[x].className;}
function setClass(x,y,cName){GameBar.children[y].children[x].className=cName;}

function moveBar()
{
    if(gameState!=1) return;
    dropLine=true;
    for(i=0;i<4;i++) if(BY[i]==15) dropLine=false;

    if(dropLine) for(i=0;i<4;i++) if(getClass(BX[i],BY[i]+1)=="SB") dropLine=false;
    if(!dropLine){
        window.clearInterval(mTimer);
        delLine();
        return;
    }
    for(i=0;i<4;i++) setClass(BX[i],BY[i],"BK");
    for(i=0;i<4;i++) BY[i]=BY[i]+1;
    for(i=0;i<4;i++) setClass(BX[i],BY[i],"MB");
}

function randBar()
{
    randNum=Math.floor(Math.random()*20)+1;
    if(!firstView)
        for(i=0;i<4;i++){
            BX[i]=PX[i];
            BY[i]=PY[i];
        }
    switch(randNum){
        case 1:
            PX[0]=4;PY[0]=0;PX[1]=4;PY[1]=1;PX[2]=5;PY[2]=1;PX[3]=6;PY[3]=1;break;
        case 2:
            PX[0]=4;PY[0]=0;PX[1]=5;PY[1]=0;PX[2]=4;PY[2]=1;PX[3]=4;PY[3]=2;break;
        case 3:
            PX[0]=4;PY[0]=0;PX[1]=5;PY[1]=0;PX[2]=6;PY[2]=0;PX[3]=6;PY[3]=1;break;
        case 4:
            PX[0]=5;PY[0]=0;PX[1]=5;PY[1]=1;PX[2]=5;PY[2]=2;PX[3]=4;PY[3]=2;break;
    }
}

```

```

case 5:
    PX[0]=6;PY[0]=0;PX[1]=6;PY[1]=1;PX[2]=4;PY[2]=1;PX[3]=5;PY[3]=1;break;
case 6:
    PX[0]=4;PY[0]=0;PX[1]=4;PY[1]=1;PX[2]=4;PY[2]=2;PX[3]=5;PY[3]=2;break;
case 7:
    PX[0]=4;PY[0]=0;PX[1]=4;PY[1]=1;PX[2]=5;PY[2]=0;PX[3]=6;PY[3]=0;break;
case 8:
    PX[0]=4;PY[0]=0;PX[1]=5;PY[1]=0;PX[2]=5;PY[2]=1;PX[3]=5;PY[3]=2;break;
case 9:
    PX[0]=4;PY[0]=0;PX[1]=5;PY[1]=0;PX[2]=5;PY[2]=1;PX[3]=6;PY[3]=1;break;
case 10:
    PX[0]=5;PY[0]=0;PX[1]=5;PY[1]=1;PX[2]=4;PY[2]=1;PX[3]=4;PY[3]=2;break;
case 11:
    PX[0]=4;PY[0]=1;PX[1]=5;PY[1]=1;PX[2]=5;PY[2]=0;PX[3]=6;PY[3]=0;break;
case 12:
    PX[0]=4;PY[0]=0;PX[1]=4;PY[1]=1;PX[2]=5;PY[2]=1;PX[3]=5;PY[3]=2;break;
case 13:
    PX[0]=4;PY[0]=0;PX[1]=5;PY[1]=0;PX[2]=6;PY[2]=0;PX[3]=5;PY[3]=1;break;
case 14:
    PX[0]=4;PY[0]=0;PX[1]=4;PY[1]=1;PX[2]=4;PY[2]=2;PX[3]=5;PY[3]=1;break;
case 15:
    PX[0]=5;PY[0]=0;PX[1]=5;PY[1]=1;PX[2]=4;PY[2]=1;PX[3]=6;PY[3]=1;break;
case 16:
    PX[0]=5;PY[0]=0;PX[1]=5;PY[1]=1;PX[2]=5;PY[2]=2;PX[3]=4;PY[3]=1;break;
case 17:
    PX[0]=4;PY[0]=0;PX[1]=5;PY[1]=0;PX[2]=4;PY[2]=1;PX[3]=5;PY[3]=1;break;
case 18:
    PX[0]=4;PY[0]=0;PX[1]=5;PY[1]=0;PX[2]=4;PY[2]=1;PX[3]=5;PY[3]=1;break;
case 19:
    PX[0]=3;PY[0]=0;PX[1]=4;PY[1]=0;PX[2]=5;PY[2]=0;PX[3]=6;PY[3]=0;break;
case 20:
    PX[0]=5;PY[0]=0;PX[1]=5;PY[1]=1;PX[2]=5;PY[2]=2;PX[3]=5;PY[3]=3;break;
}
if(firstView){
    firstView=false;
    randBar();
    return;
}
for(i=0;i<4;i++){
    for(j=0;j<4;j++){
        previewBar.children[j].children[i].className="BK";
    }
}
for(i=0;i<4;i++)previewBar.children[PY[i]].children[PX[i]-
3].className="MB";
for(i=0;i<4;i++){
    if(getClass(BX[i],BY[i])!="BK"){
        alert("Game Over!");
        window.clearInterval(mTimer);
        Play.disabled=false;
        gameState=0;
        return;
    }
}
for(i=0;i<4;i++)setClass(BX[i],BY[i],"MB");
}

function unSel()
{
    document.execCommand("Unselect");
    window.setTimeout("unSel()",10);
}
unSel();

```

```

window.onunload=rel;

function rel()
{
    location.reload();
    return false;
}

</script>

<BODY bgcolor=white onLoad="init()" onkeydown="return keyControl();">

<div id="area1" style="position:absolute;left:10px;top:10px;"></div>

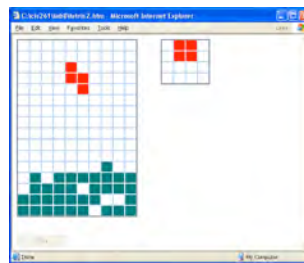
<div id="area2" style="position:absolute;left:300px;top:10px;"></div>

<button id="Play" style="position:absolute;left:10px;top:400px;width:100px"
onclick="beginGame();">Play</button>

</BODY>
</HTML>

```

3. Test the program. A sample output looks:



### Submittal

Upon completing all the learning activities,

1. Upload all files you created in this lab to your remote web server.
2. Log in to Blackboard, launch Assignment 12, and then scroll down to question 11.
3. Copy and paste the URLs to the textbox. For example,
  - [http://www.geocities.com/cis261/lab12\\_1.htm](http://www.geocities.com/cis261/lab12_1.htm)
  - [http://www.geocities.com/cis261/lab12\\_2.htm](http://www.geocities.com/cis261/lab12_2.htm)
  - [http://www.geocities.com/cis261/lab12\\_3.htm](http://www.geocities.com/cis261/lab12_3.htm)
  - [http://www.geocities.com/cis261/lab12\\_4.htm](http://www.geocities.com/cis261/lab12_4.htm)
  - [http://www.geocities.com/cis261/lab12\\_5.htm](http://www.geocities.com/cis261/lab12_5.htm)

No credit is given to broken link(s).

## Game Programming

### Lecture #13 Code Optimization

**Introduction** In computing, optimization is the process of modifying a system to make some aspect of it work more efficiently or use fewer resources. A computer game may be optimized so that it executes more rapidly, or is capable of operating with less memory storage or other resources, or draw less power.

Sometimes the biggest problem with a program is that it requires simply too many resources, and these resources (memory, CPU cycles, network bandwidth, or a combination) may be limited. Code fragments that occur multiple times throughout a program are likely to be size-sensitive, while code with many execution iterations may be speed-sensitive.

Using a better algorithm probably will yield a bigger performance increase than any amount of low-level optimizations and is more likely to deliver an improvement under all execution conditions. As a rule, high-level optimizations should be considered before doing low-level optimizations. However, before optimizing, you should carefully consider whether you need to optimize at all. According to the pioneering computer scientist Donald Knuth, "Premature optimization is the root of all evil."

**Modularization** If the program contains the same or similar blocks of statements or it is required to process the same function several times, you can avoid redundancy by using modularization techniques. By modularizing the game programs you make them easy to read and improve their structure. Modularized programs are also easier to maintain and to update. Consider the following code, in which all the objects (c1, c2, ..., c7) have some common properties, such as position, width, and height.

```
<html>
<span id="c1" style="position:relative; width:30; height:30;
background-Color:red"></span>
<span id="c2" style="position:relative; width:30; height:30;
background-Color:green"></span>
<span id="c3" style="position:relative; width:30; height:30;
background-
Color:orange"></span>
<span id="c4" style="position:relative; width:30; height:30;
background-Color:blue"></span>
<span id="c5" style="position:relative; width:30; height:30;
background-
Color:yellow"></span>
<span id="c6" style="position:relative; width:30; height:30;
background-Color:pink"></span>
<span id="c7" style="position:relative; width:30; height:30;
background-Color:black"></span>
</html>
```

Since all these properties have exactly the same values, you can extract out these common properties and make them an individual code block that can be access by each object. For example,

```
<html>

<style>
span { position:relative; width:30; height:30; }
</style>
```

```

<span id="c1" style="background-Color:red"></span>
<span id="c2" style="background-Color:green"></span>
<span id="c3" style="background-Color:orange"></span>
<span id="c4" style="background-Color:blue"></span>
<span id="c5" style="background-Color:yellow"></span>
<span id="c6" style="background-Color:pink"></span>
<span id="c7" style="background-Color:black"></span>
</html>

```

This shareable code block can be shared by as many object as you deem it necessary. It is a good way to keep the file size small. For example,

```

<html>

<style>
span { position:relative; width:30; height:30; }
</style>

<script>
var code = "";
function init() {
  for (i=0; i<=255; i++) {
    code += "<span style='background-Color:rgb(\"+i+\", \"+i+\", \"+i+\") '></span>";
  }
  area.innerHTML = code;
}
</script>

<body onLoad="init()">

<div id="area"></div>

</body>
</html>

```

Obviously creating reusable codes is a good way of modularization.

Use repetition structure to generate codes

Consider the following code segment. By properly using a repetition structure, such as the “*for*” loop, in a game script, the length of the code in a game script is significantly shortened. Thus, the client machine downloads a size-reduced game script file, uses its processing power to generate and display images on the screen without losing any game functions.

```

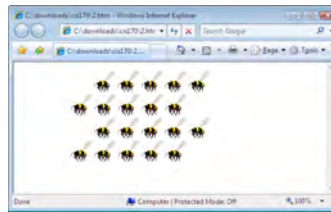
var k=1;
code="";
function init() {
  for (i=1; i<=20; i++) {

    if (i%5==1) {
      code += "<span id='s'+k+' ' style='top:'+(k-1)*40+' '>";
      code += "<img id='b'+i+' ' src='bee.gif'>";
      k++;
    }
    else if (i%5==0) { code += "<img id='b'+i+' ' src='bee.gif'></span><br>"; }
    else { code += "<img id='b'+i+' ' src='bee.gif'>"; }
  }

  area.innerHTML= code;
}

```

Notice that the number of images (e.g. 20) to be generated is technically of no limit, and such code generation is done completely at the client side. It programmatically solves the problem of file size augmentation.



Since this method is a combination of repetition structure with automatic code generation, it has a potential to allow players to decide how many objects the game has each time he/she plays the game.

Function  
lossless code  
optimization

“Lossless” is a term used in Computer Sciences to refer to a compression method in which no data is lost. A code optimization method is said to be lossless if it retains the functions without degrading any part of it.

In game programming, a game program frequently has to generate codes by itself as response to the player’s inputs. The term “**automatic code generation**” (ACG) refers to a technique programmers can use to tell computers how they generate game codes automatically according to the patterns and structures the programmer built inside the program.

Consider the following example. When a user enters a value (e.g. 7) to  $x$  (which in this case is a textbox), the variable  $y$  will be assigned a value of  $x*x$  ( $x$  squared). The value of  $y$  (e.g. 49) is then used as the terminal value of the *for* loop. The *for* loop in turn create a  $x \times x$  table as shown in the figure below.

```
<script>
function init() {
var x = eval(frm.x.value);
var y = x*x;

frm.style.display="none";

var code1="";
for (i=1; i<=y; i++) {
  if (i%x==0) {
    code1 += "<span id=c"+i+" class='cell'>"+i+"</span><br>";
  }
  else { code1 += "<span id=c"+i+" class='cell'>"+i+"</span>"; }
}
area.innerHTML = code1;
}
</script>
```

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42
43	44	45	46	47	48	49

Inside the *for* loop is a predefined code pattern and structure, as shown below. Notice that  $i$  is a value automatically assigned by the *for* loop based on whatever value  $i$  currently is, and its value

is in a range from 1 to  $y$ .

```
<span id=ci class='cell'>i</span>
```

When  $i$  is 1, the program automatically generates the following line (and that builds the first cell):

```
<span id=c1 class='cell'>1</span>
```

When  $i$  is 2, the program automatically generates the following line (and that builds the second cell):

```
<span id=c2 class='cell'>2</span>
```

When  $i$  is 3, 4, 5, ...,  $n$ , the program automatically generates the following lines:

```
<span id=c3 class='cell'>3</span>
<span id=c4 class='cell'>4</span>
<span id=c5 class='cell'>5</span>
.....
<span id=cn class='cell'>n</span>
```

The following *if* statement contains a condition  $i \% x == 0$  which defines another pattern. The % sign is not the percentage sign, it represents the **modulus** operator. Modulus is a lesser known arithmetic operator used to find the remainder of division between two numbers. For example,  $6 \div 2 = 3$  with a remainder 0 (the abbreviation is  $6/2=3r0$ ) and  $7/5 = 1r2$ . You can use the modulus operator just to get the remainder and ignore the quotient, for example,  $6\%2=0$  and  $7\%5=2$ . In other words, when a value  $i \% x == 0$ , then  $i$  must be a multiple of  $x$ .

```
if (i%x==0) {
    code1 += "<span id=c"+i+" class='cell'>"+i+"</span><br>";
} else { code1 += "<span id=c"+i+" class='cell'>"+i+"</span>"; }
```

The above code uses the condition  $i \% x == 0$  to decide if the current value of  $i$  is a multiple of  $x$ . If  $i$  is a multiple of  $x$ , add an **<br>** tag as shown below:

```
<span id=c1 class='cell'>1</span><br>
```

When  $x=7$ , then each time when  $i$  is a multiple of 7 (e.g. 7, 14, 21, 28, ...49), the **<br>** tag will break a line, so the next number starts a new line. Namely, 1 through 7 make the first line, 8 through 13 make the second line, 14 through 20 make the third line, and so on.

```
.....
<span id=c6 class='cell'>6</span>
<span id=c7 class='cell'>7</span><br>
.....
<span id=c13 class='cell'>13</span>
<span id=c14 class='cell'>14</span><br>
.....
<span id=c20 class='cell'>20</span>
<span id=c21 class='cell'>21</span><br>
.....
```

ACG (automatic code generation) is also commonly used to automatically insert new objects to a game. Consider the following code, which continuously adds color bricks (in red, green, or blue) randomly to the screen.

```
<html>
```

```

<style>
span { position: absolute; width:25px;height:25px }
</style>

<script>
var i=0;
var s,t, u, v, x, y;
var code=""
function init() {

s = Math.floor(Math.random()*document.body.clientWidth/25);
t = Math.floor(Math.random()*document.body.clientHeight/25);

u = Math.floor(Math.random()*document.body.clientWidth/25);
v = Math.floor(Math.random()*document.body.clientHeight/25);

x = Math.floor(Math.random()*document.body.clientWidth/25);
y = Math.floor(Math.random()*document.body.clientHeight/25);

code+"<span id=r"+i+" style='left:"+s*25+";top:"+t*25+";background-Color:red;'></span>"

code+="<span id=g"+i+" style='left:"+u*25+";top:"+v*25+";background-Color:green;'></span>"

code+="<span id=b"+i+" style='left:"+x*25+";top:"+y*25+";background-Color:blue;'></span>"

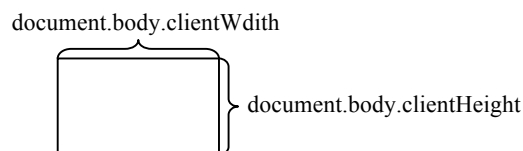
bd.innerHTML += code;
i++;
setTimeout("init()", 5);
}
</script>

<body onLoad="init()">
<div id="bd"></div>
</body>

</html>

```

The current width of web browser is represented by **document.body.clientWidth**, while the current height of the web browser is represented by **document.body.clientHeight**. Since the width and height of each brick is set to be  $25 \times 25$  (pixels).



The following will determine the maximum number of brick per row and per column allows.

```

document.body.clientWidth/25
document.body.clientHeight/25

```

The following will randomly specify a value that represents the x-coordinate of a brick.

```

Math.floor(Math.random()*document.body.clientWidth/25);

```

Similarly, the following will randomly specify a value that represents the y-coordinate of a brick.

```
Math.floor(Math.random()*document.body.clientHeight/25);
```

You then use the randomly generate  $(x, y)$  values to one red, one green, and one blue brick each time when the **init()** functions is called. This part of code is:

```
code=<span id=r"+i+" style='left:"+s*25+";top:"+t*25+";background-
Color:red;'></span>"

code+="

```

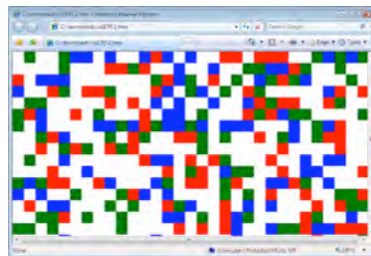
The innerHTML property will then insert the newly generated codes a division with an ID **bd**.

```
bd.innerHTML += code;
```

The **i++** means to add 1 to the current value of *i*, so the value of *i* is incremented by 1 each time when this statement is read by computer. The **setTimeout()** method force the computer to call the **init()** function every 5 milliseconds

```
setTimeout("init()", 5);
```

A sample output looks:



Colorful bricks will be randomly inserted to blank space.

Clearing  
physical  
memory

Many game codes create objects that have limited life span (duration). For example a bullet that the player uses to shoot the enemy. When the bullet is fired, its life span should soon end if the bullet miss or hit the target. Since each object in the game takes up certain memory space, it is always a good practice to clear unneeded, unnecessary or unwanted object from the physical memory.

Consider the following code, which does not delete the unneeded object from memory.

```
<html>

<style>
span {position:absolute;}
</style>

<script>

function init() {
pl.innerHTML = "<span id='c'>o</span>";
move();
}

}
```

```

function move() {
  if (c.style.pixelLeft >=200) {
    clearTimeout(s1); p1.innerText = '';}
  else { c.style.pixelLeft++;
    s1=setTimeout('move()', 10);}
}

</script>

<body onKeyDown="init();">

<b id="p1"></b>
</body>
</html>

```

This game code functions well, but the only problem is. Each time when the player press a key, the “c” object is created again without deleting the previously created “c” object. Consequently, after pressing a key many times, the player will encountered some problem caused by memory overuse.

Applying object-oriented programming technique, in this case, helps to remove unneeded objects. JavaScript support object-oriented programming, so you can create an object, use the object for a period of time, and delete the object later.

```

<html>

<style>
span {position:absolute;}
</style>

<script>
var i = 1;

function init() {
obj = new Object();
obj.n="<span id='c'+i+'>o</span>";
p1.innerHTML = obj.n;
move();
}

function move() {
code = "if (c"+i+".style.pixelLeft >=200) { ";
code += " clearTimeout(s1); obj = '';p1.innerText = ''; i++;}";
code += "else { c"+i+".style.pixelLeft++; status=i; ";
code += " s1=setTimeout('move()', 10);}";
eval(code);
}

</script>

<body onKeyDown="init();">

<b id="p1"></b>
</body>
</html>

```

To create an object, assign an object ID (e.g. obj) to the Object() method.

```
obj = new Object();
```

You can then, create properties for the **obj** object. In the following example, “text” is a new property of the obj object with a string value “Test the code”.

```
obj.test = "Test the code";
```

To delete the object, simply assign the object ID with null value;

```
obj = '';
```

Once the object is deleted, the game codes that control it are no longer kept in physical memory.

Execution  
issues and  
speed

In a previous lecture, you tried the UFO game, whose source code looks:

```
<html>
<style>
.dots {position:absolute; color:white}
</style>

<script>

code1 = "";
function draw_star() {
  for (i=1; i<=600; i++) {
    x = Math.round(Math.random()*screen.width);
    y = Math.round(Math.random()*screen.height);
    code1 += "<span class=dots id=dot"+i;
    code1 += " style='left:" + x + "; top:" + y + "'>.</span>";
  }
  bar.innerHTML = code1;
}

function ufo_fly() {
  ufo.style.left = Math.round(Math.random()*screen.width);
  ufo.style.top = Math.round(Math.random()*screen.height);
  setTimeout("ufo_fly()", 500);
}

</script>

<body bgcolor=black onLoad="draw_star();ufo_fly()">
<div id="bar" style="z-index:0"></div>


</body>
</html>
```

If you modify just the **draw\_star()** function to the following, the loading time will be much longer, and the execution will be slower.

```
<script>

function draw_star() {
  for (i=1; i<=300; i++) {
    x = Math.round(Math.random()*screen.width);
    y = Math.round(Math.random()*screen.height);

    code1 = "<span class=dots id=dot"+i;
    code1 += " style='left:" + x + "; top:" + y + "'>.</span>";
    bar.innerHTML += code1;
  }
}
```

```
.....
</script>
```

Try take a closer look at these two code segments:

<pre>code1 = ""; function draw_star() {   for (i=1; i&lt;=600; i++) {     x = Math.round(Math.random()*screen.width);     y = Math.round(Math.random()*screen.height);     code1 += "&lt;span class=dots id=dot"+i;     code1 += " style='left:" + x + "; top:" + y + "'&gt;.&lt;/span&gt;";   }   bar.innerHTML = code1; // outside the for loop }</pre>
<pre>function draw_star() {   for (i=1; i&lt;=300; i++) {     x = Math.round(Math.random()*screen.width);     y = Math.round(Math.random()*screen.height);      code1 = "&lt;span class=dots id=dot"+i;     code1 += " style='left:" + x + "; top:" + y + "'&gt;.&lt;/span&gt;";     bar.innerHTML += code1; // inside the for loop   } }</pre>

The trick is that in the upper one, the following line is placed outside the *for* loop. But, in the lower one, it is place insider the *for* loop.

```
bar.innerHTML = code1;
```

When being executed, the one outside the *for* loop, does not have to insert the HTML code till the entire *for* loop completes. Each time the a new line, similar to the following, is produced by the *for* loop, they are temporarily stored in the computer's memory (buffer) as additional one to the previous ones.

```
<span class=dots id=dotn style='left: x; top: y'>.</span>
```

Where *n* is one of the number in {1, 2, 3, ..., 300}, while *x*, *y* and randomly generated by the *draw\_star()* function.

At the end of the *for* loop, the computer memory accumulates the following new HTML codes, and insert all them to the area defined by `<div id="bar" style="z-index:0"></div>`.

```
<span class=dots id=dot1 style='left: x1; top: y1'>.</span>
<span class=dots id=dot2 style='left: x2; top: y2'>.</span>
<span class=dots id=dot3 style='left: x3; top: y3'>.</span>
.....
<span class=dots id=dot300 style='left: x300; top: y300'>.</span>
```

It is faster to insert all the above lines to the web page, then inserting then one by one, which is exactly what the second code segement has to do. When you place the following line inside the *for* loop, the processing mechanism becomes redundant.

```
bar.innerHTML += code1;
```

First of all, each time when a *for* loop starts, the computer needs to go to the `<div id="bar"`

...></div> area to read the currently available lines of code similar to following, and then insert a new one to it:

```
<span class=dots id=dotn style='left: x; top: y'>.</span>
```

It takes a significantly longer time to find the spot, read the contents, and add new contents to it. Plus, the computer has to do it 300 times. No wonder why the execution time is longer, and the speed is significantly slower.

#### Review Questions

1. \_\_\_ is the process of modifying a system to make some aspect of it work more efficiently or use fewer resources.  
A. Optimization  
B. File size reduction  
C. Performance tuning  
D. Finetuning

2. If the program contains the same or similar blocks of statements or it is required to process the same function several times, you can avoid redundancy by using \_\_\_ techniques.  
A. reengineering  
B. collaboration  
C. modularization  
D. packaging

3. The following is an example of \_\_\_.

```
<style>span { position:relative; width:30; height:30; background-Color:red}
</style>

<span id="c1"></span>
<span id="c2"></span>
<span id="c3"></span>
<span id="c4"></span>
<span id="c5"></span>
<span id="c6"></span>
<span id="c7"></span>
```

- A. automatic code generation
- B. modularization
- C. object specialization
- D. object generalization

4. By properly using a repetition structure, such as the “for” loop, in a game script, \_\_\_\_.  
A. the length of the code in a game script is significantly shortened.  
B. the client machine downloads a size-reduced game script file.  
C. the client machine uses its processing power to generate and display images on the screen without losing any game functions.  
D. All of the above.

5. The following is an example of \_\_\_\_.

```
<script>
function init() {
var x = eval(frm.x.value);
var y = x*x;

frm.style.display="none";
```

```

var code1="";
for (i=1; i<=y; i++) {
  if (i%x==0) {
    code1 += "<span id=c"+i+" class='cell'>"+i+"</span><br>";
  }
  else { code1 += "<span id=c"+i+" class='cell'>"+i+"</span>"; }
}
area.innerHTML = code1;
}
</script>

```

- A. automatic code generation
- B. modularization
- C. object specialization
- D. object generalization

6. In the following code, the condition `i%x==0` \_\_.

```

if (i%x==0) { ..... }

```

- A. decide if i is a percentage of x
- B. determine if i equals x
- C. decide if the current value of i is a multiple of x
- D. determine if i equals 0

7. Given the following code segment, what happens when `i=14` and `x=7`?

```

if (i%x==0) {
  code1 += "<span id=c"+i+" class='cell'>"+i+"</span><br>";
}
else { code1 += "<span id=c"+i+" class='cell'>"+i+"</span>"; }
}

```

- A. Since i is a multiple of x, the output is `<span id=c14 class='cell'>14</span><br>`.
- B. Since i is a multiple of x, the output is `<span id=c14 class='cell'>14</span>`.
- C. Since i is a multiple of x, the output is `<span id=c7 class='cell'>7</span><br>`.
- D. Since i is a multiple of x, the output is `<span id=c7 class='cell'>7</span>`.

8. The following line \_\_.

```

m1 = new Object();

```

- A. assign the variable m1 with a new value "Object()".
- B. create a new object with an ID m1.
- C. for the computer to ignore the m1 object.
- D. delete the m1 object from the current program.

9. In JavaScript, you can delete an object by \_\_.

- A. assigning a new value to it.
- B. re-declaring the object.
- C. assining a null value to it.
- D. converting the object to a variable.

10. Given the following incomplete code segemetn, where will you insert the `bar.innerHTML = code;` statement for the best execution performance?

```

.....
code = "";
function init() {
  // area1

```

```
for (i=1; i< 10; i++) {  
    code += "<hr size="+i+" width=100%>";  
    // area2  
}  
// area3  
}  
// area4  
.....  
<div id="bar"></div>  
.....
```

- A. area1
- B. area2
- C. area3
- D. area4

## Game Programming

### Lab #13

### Code Optimization

#### Learning Activity #1:

1. Use Notepad to create a new file named lab13\_1.htm with the following contents:

```
<html>

<style>
span {position:absolute;}
</style>

<script>
var i = 1;

function init() {
obj = new Object();
obj.n="<span id='c'+i+'>o</span>";
p1.innerHTML = obj.n;
move();

}

function move() {
code = "if (c"+i+".style.pixelLeft >=200) { ";
code += " clearTimeout(s1); obj = '';p1.innerHTML = ' '; i++;}";
code += "else { c"+i+".style.pixelLeft++; status=i; ";
code += " s1=setTimeout('move()', 10);}";
eval(code);

}

</script>

<body onKeyDown="init();">

<b id="p1"></b>
</body>
</html>
```

2. Use Internet Explorer to test the program. There will be a continuously changing number displayed in the status bar.



#### Learning Activity #2:

1. Use Notepad to create a new file named lab13\_2.htm with the following contents:

```
<html>

<style>
.cell {
```



```

t = Math.floor(Math.random()*document.body.clientHeight);

x = Math.floor(Math.random()*document.body.clientWidth);
y = Math.floor(Math.random()*document.body.clientHeight);

code="<span id=r"+i+" style='left:"+s+";top:"+t+";background-color:red;'></span>"

code+="<span id=b"+i+" style='left:"+x+";top:"+y+";background-color:blue;'></span>"

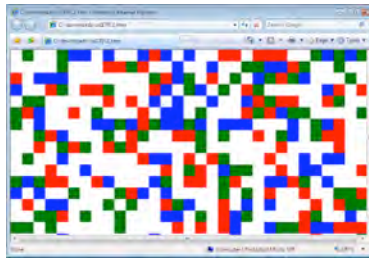
bd.innerHTML += code;
i++;
setTimeout("init()", 5);
}
</script>

<body onload="init()">
<div id="bd"></div>
</body>

</html>

```

2. Use Internet Explorer to test the program. A sample output looks:



#### Learning Activity #4:

1. Use Internet Explorer to go to <http://business.cypresscollege.edu/~pwu/cis261/files/bee.gif> and download the bee.gif file.
2. Use Notepad to create a new file named lab13\_4.htm with the following contents:

```

<html>

<style>
img { position:relative; }
span {position:absolute;}
</style>

<script>
var k=1;
code="";
function init() {
  for (i=1; i<=20; i++) {

    if (i%5==1) { code += "<span id='s"+k+"' style='top:"+ (k-1)*40+"'"><img
id='b"+i+"' src='bee.gif'">"; k++; }
    else if (i%5==0) { code += "<img id='b"+i+"' src='bee.gif'"></span><br>"; }
    else { code += "<img id='b"+i+"' src='bee.gif'">"; }
  }

  area.innerHTML= code;

  for (j=1; j<=4; j++) {
    eval("s"+j+".style.pixelWidth = 250;");
  }
}

```

```

movebee();
}

function movebee() {

    if (s1.style.pixelLeft + s1.style.pixelWidth >= document.body.clientWidth) {
        s1.style.pixelLeft = 0
    }

    else { s1.style.pixelLeft += 5; }

    if (s2.style.pixelLeft + s2.style.pixelWidth >= document.body.clientWidth) {
        s2.style.pixelLeft = 0
    }

    else {
        if (s1.style.pixelLeft<=40) { s2.style.pixelLeft =0; }
        else { s2.style.pixelLeft += 5; }
    }

    if (s3.style.pixelLeft + s3.style.pixelWidth >= document.body.clientWidth) {
        s3.style.pixelLeft = 0
    }

    else { s3.style.pixelLeft += 5; }

    if (s4.style.pixelLeft + s4.style.pixelWidth >= document.body.clientWidth) {
        s4.style.pixelLeft = 0
    }

    else {
        if (s3.style.pixelLeft<=40) { s4.style.pixelLeft =0; }
        else { s4.style.pixelLeft += 5; }
    }

    setTimeout("movebee()", 100);
}

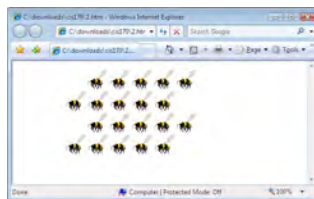
</script>

<body onLoad="init()">
<div id="area" style="position:absolute"></div>
</body>

</html>

```

3. Use Internet Explorer to test the program.



#### Learning Activity #5:

Note: This code is a fully functional game code, but it will function better if you apply the object-oriented programming techniques (as discussed in the lecture) to it. Try modify the code with object-oriented programming.

1. Use Notepad to create a new file named lab13\_5.htm with the following contents:

```
<html>

<script>
var i=0;
function init() {
  if (red.style.pixelLeft >= document.body.clientWidth) {
    red.style.pixelLeft=0; red.style.pixelTop+=20;
  }
  else if (red.style.pixelTop >= document.body.clientHeight-20) {
    red.style.pixelTop=0;
  }
  red.style.pixelLeft+=5;

  for (k=1; k<=i; k++) {
    eval("b"+i+".style.pixelTop+=5");

    msg1="if (b"+i+".style.pixelTop >= document.body.clientHeight-100)";
    msg1+=" {b"+i+".style.display='none';}";
    eval(msg1);

    msg2="if (b"+i+".style.pixelLeft>=red.style.pixelLeft &&";
    msg2+=" b"+i+".style.pixelLeft<=red.style.pixelLeft+20 &&";
    msg2+=" b"+i+".style.pixelTop+20>=red.style.pixelTop &&";
    msg2+=" b"+i+".style.pixelTop<=red.style.pixelTop+20) ";
    msg2+="{ red.style.display='none';p1.innerText='Game Over!';}";
    eval(msg2);
  }

  setTimeout("init()", 20);
}

function fire() {
  i=i+1;
  b_top=Math.floor(Math.random()*document.body.clientWidth);

  code="<span id='b"+i+"' style='position:absolute; ";
  code+=" background-Color:blue; width:20; ";
  code+=" height:20; top:0; left:"+b_top+"'></span> ";

  bd.innerHTML+=code;
  eval("b"+(i-1)+" style.display='none';");
}
</script>

<body id="bd" onLoad="init()" onKeyDown="var f = new fire()"
  style="overflow: hidden">
<span id="red" style="position:absolute; background-Color:red; width:20;
  height:20; top:100"></span>

<p id="p1"></p>
</body>
</html>
```

2. Use Internet Explorer to test the program.

### Submittal

Upon completing all the learning activities,

1. Upload all files you created in this lab to your remote web server.
2. Log in to Blackboard, launch Assignment 13, and then scroll down to question 11.
3. Copy and paste the URLs to the textbox. For example,
  - [http://www.geocities.com/cis261/lab13\\_1.htm](http://www.geocities.com/cis261/lab13_1.htm)
  - [http://www.geocities.com/cis261/lab13\\_2.htm](http://www.geocities.com/cis261/lab13_2.htm)
  - [http://www.geocities.com/cis261/lab13\\_3.htm](http://www.geocities.com/cis261/lab13_3.htm)
  - [http://www.geocities.com/cis261/lab13\\_4.htm](http://www.geocities.com/cis261/lab13_4.htm)
  - [http://www.geocities.com/cis261/lab13\\_5.htm](http://www.geocities.com/cis261/lab13_5.htm)

No credit is given to broken link(s).

## Game Programming

### Lecture #14 Using Canvas and ExplorerCanvas

**Introduction** The **canvas** element is part of HTML5 and allows for dynamic scriptable rendering of bitmap images. Canvas consists of a drawable region defined in HTML code with height and width attributes. JavaScript code may access the area through a full set of drawing functions similar to other common 2D APIs, thus allowing for dynamically generated graphics. Some anticipated uses of the canvas include building graphs, animations, games, and image composition.

Firefox, Safari and Opera 9 support the canvas tag by default, but Internet Explorer does not. Thanks to Google's efforts in creating the ExplorerCanvas, which seems to solve this problem to some extent.

ExplorerCanvas brings the same functionality to Internet Explorer. To use, web developers only need to include a single script tag in their existing web pages.

Good news is that Google distributes ExplorerCanvas with the Open Source **BSD License** and the **Apache License**. This is free license that allows use of the source code for the development of free and open source software as well as proprietary software.

Sample codes in this lecture note that use ExplorerCanvas respect Google's copyright by adding the following line at the beginning of the HTML codes.

```
<!--This code uses ExplorerCanvas, developed by Google Inc.-->
```

**Using ExplorerCanvas** Simply include the ExplorerCanvas tag in the same directory as your HTML files, and add the following code to your page, preferably in the <head> tag.

```
<!--[if IE]>
<script type="text/javascript" src="excanvas.js"></script>
<![endif]-->
```

For example,

```
<!--This code uses ExplorerCanvas, developed by Google Inc.-->

<html><head>

<!--[if IE]><script type="text/javascript"
src="excanvas.js"></script><![endif]-->

<script type="text/javascript">
function init() {
    var sid = document.getElementById("sqr");

    if (sid.getContext) {
        var obj = sid.getContext("2d");

        obj.fillStyle = "rgb(255, 0, 0)";
        obj.fillRect (10, 10, 100, 100);
    }
}
</script></head>

<body onload="init();">
<canvas id="sqr" width="150" height="150"></canvas>
</body></html>
```

The <canvas> element

The <canvas> element creates a fixed size drawing surface that can render graphics. It has only three attributes, **id**, **width** and **height**. These are both optional and can also be set using DOM properties. When no width and height attributes are specified, the canvas will initially be 300 pixels wide and 150 pixels high. To avoid compatibility issues, be sure to have an end tag </canvas>.

```
<canvas id="sqr" width="150" height="150"></canvas>
```

The **id** attribute is used to identify the <canvas> tag so it can be used as an object to work with the Document Object Model (DOM) API.

The <canvas> element also works with CSS, so it can be styled with margin, border, background color, etc. For example,

```
<style type="text/css">
  canvas { border: 1px solid black; }
</style>
```

The **getContext()** method is a DOM method that retrieves the current active document type and document ID. It works with the **getElementById** method to identify the object that will serve as the drawing ground. For example,

```
var sid = document.getElementById('sqr');
.....
var obj = sid.getContext('2d');
.....
<canvas id="sqr" width="150" height="150"></canvas>
```

In the above example, “sqr” is the ID of the <canvas>, which is the drawing ground. The **getElementById** method retrieve the ID (sqr) and transfers it a variable **sid**. The sid variable is then used as a object to retrieve drawing context that will be specified by getContext() method.

You can combine the following two lines,

```
var sid = document.getElementById('sqr');
var obj = sid.getContext('2d');
```

to

```
var obj = document.getElementById('sqr').getContext('2d');
```

There are still some browser that do not support the <canvas> element. You may want to display some message notifying the viewer such incompatibility. A simple if..then..else statement can do the work. For example,

```
var sid = document.getElementById('sqr');
if (sid.getContext){
  var obj = canvas.getContext('2d');
  // drawing code here
} else {
  // message saying canvas unsupported here
}
```

The drawing codes

The drawing of the graphics is done by drawing functions. Commonly used drawing functions are:

Table: Drawing functions

Function	Description
----------	-------------

<code>fillRect(x,y,width,height)</code>	Draws a filled rectangle
<code>strokeRect(x,y,width,height)</code>	Draws a rectangular outline
<code>clearRect(x,y,width,height)</code>	Clears the specified area and makes it fully transparent
<code>rect(x, y, width, height)</code>	Adds a rectangular path to the path list

where *x* and *y* specify the position on the canvas (relative to the origin) of the top-left corner of the rectangle. *width* and *height* are pretty obvious.

Drawing function only handler the sketching. If you want to apply colors to a shape, there are two important properties to use: **fillStyle** and **strokeStyle**. Basically **strokeStyle** is used for setting the shape outline color and **fillStyle** is for the fill color. Color codes can be a string representing a CSS color value, a gradient object, or a pattern object. We'll look at gradient and pattern objects later. By default, the stroke and fill color are set to black (CSS color value #000000). For example,

```
obj.fillStyle = "orange";
obj.fillStyle = "#FFA500";
obj.fillStyle = "rgb(255,165,0)";
obj.fillStyle = "rgba(255,165,0,1)";
```

All the above generates exactly the same color.

The `rgb()` function has syntax of:

```
rgb(redvalue, greeValue, blueValue);
```

Each color value is between 0 and 255.

The **rgba()** function is similar to the **rgb()** function but it has one extra parameter.

```
rgba(redvalue, greeValue, blueValue, transparencyValue);
```

The last parameter of `rgba()` sets the transparency value of this particular color. The valid range is from 0.0 (fully transparent) to 1.0 (fully opaque).

To draw an rectangle with red color, use:

```
obj.fillStyle = "rgb(255, 0, 0)"; // define color
obj.fillRect (10, 10, 100, 100); // sketching
```

Currently canvas only supports one primitive shape - **rectangles**. All other shapes must be created by combining one or more paths.

The drawing path

Shapes, such as polygon, are sketched using drawing path with some of the following functions.

- **beginPath()**: creates a path
- **closePath()**: close the shape by drawing a straight line from the current point to the start
- **stroke()**: draw an outlined shape
- **fill()**: paint a solid shape

When calling the `fill()` method any open shapes will be closed automatically and it isn't necessary to use the `closePath()` method. The following code, for example, will not formed a closed rectangle.

```
obj.beginPath();
obj.moveTo(10,50);
obj.lineTo(10,10);
```

```
obj.lineTo(110,10);
obj.lineTo(110,50);
obj.stroke();
```

The output looks like:



The following code draws a triangle with filled color in red.

```
<!--This code uses ExplorerCanvas, developed by Google Inc.-->

<html><head>

<!--[if IE]><script type="text/javascript"
src="excanvas.js"></script><![endif]-->

<script type="text/javascript">
function init() {
  var sid = document.getElementById("sqr");

  if (sid.getContext) {
    var obj = sid.getContext("2d");

    obj.fillStyle = "rgb(255, 0, 0)";
    obj.beginPath();
    obj.moveTo(100,10);
    obj.lineTo(50,150);
    obj.lineTo(100,200);
    obj.fill();
  }
}
</script></head>

<body onload="init();">
  <canvas id="sqr" width="600" height="200"></canvas>
</body></html>
```

One important thing to know is that the `stroke()` function always display the line in black, so the **fillStyle** project in the following code will not display the lines in red.

```
obj.fillStyle = "rgb(255, 0, 0)";
obj.beginPath();
obj.moveTo(100,10);
obj.lineTo(50,150);
obj.lineTo(100,200);
obj.stroke();
```

When the canvas is initialized or the `beginPath` method is called, the starting point is defaulted to the coordinate (0,0). The **moveTo()** function defines the starting point. You can probably best think of this as lifting a pen or pencil from one spot on a piece of paper and placing it on the next. The syntax is:

```
moveTo(x, y)
```

where x and y are the coordinates of the new starting point.

The `lineTo` method draws straight lines, and the syntax is:

```
lineTo(x, y)
```

where x and y are the coordinates of the line's end point.

The following draws two triangles.

```
obj.beginPath();  
obj.moveTo(100,10);  
obj.lineTo(50,150);  
obj.lineTo(150,50);  
obj.lineTo(100,200);  
obj.fill();
```



The **arc()** method draws arcs or circles. The syntax is:

```
arc(x, y, radius, startAngle, endAngle, anticlockwise);
```

where x and y are the coordinates of the circle's center. Radius is self explanatory. The startAngle and endAngle parameters define the start and end points of the arc in radians. The starting and closing angle are measured from the x axis. The anticlockwise parameter is a boolean value which when true draws the arc anticlockwise, otherwise in a clockwise direction.

For example, to draw an arc with a center (100, 100) and a radius of 50, use:

```
<!--This code uses ExplorerCanvas, developed by Google Inc.-->  
  
<html><head>  
  
<!--[if IE]><script type="text/javascript"  
src="excanvas.js"></script><![endif]-->  
  
<script type="text/javascript">  
  
var i = 0;  
  
function init() {  
  var sid = document.getElementById("sqr");  
  
  if (sid.getContext) {  
    var obj = sid.getContext("2d");  
  
    obj.arc(100, 100, 50, 2 , Math.PI+(Math.PI*i)/2, true);  
    obj.stroke();  
  
  }  
  i+=0.1;  
}  
</script></head>  
  
<body onload="init();" onKeyDown="init()">  
  <canvas id="sqr" width="600"  
height="600"></canvas>  
</body></html>
```



Angles in the arc function are measured in radians, not degrees. To convert degrees to radians you can use the following JavaScript expression:

```
var radians = (Math.PI/180)*degrees
```

Review  
Questions

1. Which statement is correct?
- A. The canvas element is part of HTML5 and allows for dynamic scriptable rendering of bitmap images.
  - B. The canvas element consists of a drawable region defined in HTML code with height and width attributes.
  - C. Internet Explorer 6 and 7 do not support the canvas element.
  - D. All of the above.

2. Google provides the \_\_ library that brings the canvas element to Internet Explorer.
- A. iCanvas
  - B. J3DJS
  - C. JSON
  - D. AJAX

3. Given the following code, which statement is correct?

```
obj.fillRect(10, 10, 100, 100);
```

- A. The top-left corner of the rectangle is (100, 100).
  - B. The top-left corner of the rectangle is (10, 10).
  - C. It generates a square that has width and height of 10 pixels each.
  - D. It generates a rectangle that has a width of 10 pixels and height of 100 pixels.
4. Which defines a drawing ground that has a width of 400 pixels and height of 300 pixels?
- A. `<canvas id="sqr"><td width="400" height="300"></td></canvas>`
  - B. `<canvas id="sqr" size="300*400"></canvas>`
  - C. `<canvas id="sqr" width="400" height="300"></canvas>`
  - D. `<canvas id="sqr" w="400" h="300"></canvas>`

5. Which function draws a rectangular outline?

- A. `fillRect()`
- B. `strokeRect()`
- C. `clearRect()`
- D. `rect()`

6. Given the following code with error in it, how will you correct the error?

```
obj.fillStyle = "rgba(255,105,10,2)";
```

- A. Change `rgba` to `rgb` and keep the rest.
- B. remove 2 and the comma before it; keep the rest.
- C. change 2 to any value in the range from 0.0 to 1.0.
- D. add the # sign before every number.

7. Given the following code segment, the color of line is \_\_.

```
obj.fillStyle = "rgb(255, 0, 0)";  
obj.beginPath();  
obj.moveTo(100,10);  
obj.lineTo(50,150);  
obj.stroke();
```

- A. red
- B. black
- C. while
- D. orange

8. Given the following code, \_\_.

```
obj.beginPath();  
obj.moveTo(10,50);  
obj.lineTo(10,10);  
obj.lineTo(110,10);  
obj.lineTo(110,50);  
obj.stroke();
```

- A. It creates a closed rectangle.
- B. It creates a closed triangle.
- C. It creates a closed square.
- D. None of the above.

9. Given the following code, \_\_.

```
obj.arc(100, 100, 50, 2, Math.PI+(Math.PI*i)/2, true);
```

- A. the center of the arc is (50, 2).
- B. the radius is 50.
- C. the starting point is (100, 100).
- D. the ending point is (100, 50).

10. To convert degrees to radians you can use the following JavaScript expression:

- A. `var radians = (Math.PI/180)*degrees`
- B. `var radians = Math.PI+(Math.PI*180)/2`
- C. `var radians = (Math.PI/360)*degrees`
- D. `var radians = Math.PI+(Math.PI*360)/2`

## Game Programming

### Lab #14

### Using Canvas and ExplorerCanvas

#### Preparation #1:

1. Create a new directory c:\cis261 if it does not exist.
2. Use Internet Explorer to go to <http://business.cypresscollege.edu/~pwu/cis261/files/lab14.zip> to download lab14.zip to the c:\cis261 directory. Extract the excanvas.js file to C:\cis261 directory.

#### Learning Activity #1:

1. Change to the C:\cis261 directory.
2. Use Notepad to create a new file named C:\cis261\lab14\_1.htm, with the following:

```
<!-- This code uses ExplorerCanvas, developed by Google Inc. -->

<html><head>

<!--[if IE]><script type="text/javascript" src="excanvas.js"></script><![endif]-->

<script type="text/javascript">
var x=100;

function init() {
    var sid = document.getElementById("sqr");

    if (sid.getContext) {
        var obj = sid.getContext("2d");

        obj.fillStyle = "rgb(255, 0, 0)";
        obj.beginPath();
        obj.moveTo(100,10);
        obj.lineTo(x,150);
        obj.lineTo(100,200);
        obj.fill();
    }

    if (x<=10) { window.location=window.location; }
    x--;

    setTimeout("init()", 10);

}
</script></head>

<body onload="init();">
    <canvas id="sqr" width="600" height="200"></canvas>
</body></html>
```

3. Test the program. A triangle will grow in size.

#### Learning Activity #2

1. Use Notepad to create a new file named C:\cis261\lab14\_2.htm, with the following:

```
<!-- This code uses ExplorerCanvas, developed by Google Inc. -->

<html><head>

<!--[if IE]><script type="text/javascript" src="excanvas.js"></script><![endif]-->

<script type="text/javascript">
```

```

var y=200;

function init() {
    var sid = document.getElementById("sqr");

    if (sid.getContext) {
        var obj = sid.getContext("2d");

        obj.beginPath();
        obj.moveTo(10,200);
        obj.lineTo(10,10);
        obj.lineTo(110,10);
        obj.lineTo(110,200);
        obj.stroke();

        obj.beginPath();
        obj.moveTo(10,y);
        obj.lineTo(110,y);
        obj.stroke();
    }

    if (y <= 10) { y=10; clearTimeout(s1);}
    else { y--; }
    s1 = setTimeout("init()", 10);

}
</script></head>

<body onload="init();">
    <canvas id="sqr" width="600" height="200"></canvas>
</body></html>

```

2. Test the program. A sample output looks:



### Learning Activity #3

1. Use Notepad to create a new file named C:\cis261\lab14\_3.htm, with the following:

```

<!-- This code uses ExplorerCanvas, developed by Google Inc. -->

<html><head>

<!--[if IE]><script type="text/javascript" src="excanvas.js"></script><![endif]-->

<script type="text/javascript">

var i = 1;

function init() {
    var sid = document.getElementById("sqr");

    if (sid.getContext) {
        var obj = sid.getContext("2d");

```

```

    obj.arc(100, 100, 50, 0 , (Math.PI/180)*i, false);
    obj.stroke();

}

if (i>=360) { window.location=window.location; }

i++;
setTimeout("init()", 10);
}
</script></head>

<body onload="init();">
  <canvas id="sqr" width="600" height="600"></canvas>
</body></html>

```

2. Test the program. Watch a small arc grow into a circle.

#### Learning Activity #4

1. Use Notepad to create a new file named C:\cis261\lab14\_4.htm, with the following:

```

<!-- This code uses ExplorerCanvas, developed by Google Inc. -->

<html><head>

<!--[if IE]><script type="text/javascript" src="excanvas.js"></script><![endif]-->

<script type="text/javascript">
function init() {
  var sid = document.getElementById("sqr");

  if (sid.getContext) {
    var obj = sid.getContext("2d");

    obj.fillStyle = "rgb(255, 0, 0)";
    for (i=0;i<4;i++){
      for(j=0;j<3;j++){
        obj.beginPath();
        var x          = 25+j*50;           // x coordinate
        var y          = 25+i*50;           // y coordinate
        var radius      = 20;                // Arc radius
        var startAngle  = 0;                 // Starting point on circle
        var endAngle    = Math.PI+(Math.PI*j)/2; // End point on circle
        var anticlockwise = i%2==0 ? false : true; // clockwise or anticlockwise

        obj.arc(x,y,radius,startAngle,endAngle, anticlockwise);

        if (i>1){
          obj.fill();
        } else {
          obj.stroke();
        }
      }
    }
  }
}

</script></head>

<body onload="init();">
  <canvas id="sqr" width="600" height="200"></canvas>

```

```
</body></html>
```

2. Test the program. A sample output looks:



#### Learning Activity #5:

1. Use Notepad to create a new file named C:\cis261\lab14\_5.htm, with the following:

```
<!-- This code uses ExplorerCanvas, developed by Google Inc. -->

<html>
<meta http-equiv="refresh" content="1">

<head>

<!--[if IE]><script type="text/javascript" src="excanvas.js"></script><![endif]-->

<script type="text/javascript">

var k = Math.floor(Math.random()*50);

function init() {

    var obj = document.getElementById('sqr').getContext('2d');
    obj.translate(75,75);

    for (i=1;i<8;i++){ // Loop through rings (from inside to out)
        obj.save();
        obj.fillStyle = 'rgb('+(k*i)+','+(255-k*i)+','+(255-k*i)+')';

        for (j=0;j<i*6;j++){ // draw individual dots
            obj.rotate(Math.PI*2/(i*6));
            obj.beginPath();
            obj.arc(0,i*9.5,5,0,Math.PI*2,true);
            obj.fill();
        }

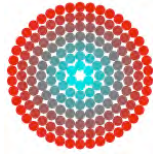
        obj.restore();
    }

}

</script></head>

<body onload="init();" onKeyDown="init()">
    <canvas id="sqr"></canvas>
</body></html>
```

2. Test the program. A sample output looks:

**Submittal**

Upon completing all the learning activities,

1. Upload all files you created in this lab to your remote web server.
2. Log in to Blackboard, launch Assignment 14, and then scroll down to question 11.
3. Copy and paste the URLs to the textbox. For example,
  - [http://www.geocities.com/cis261/lab14\\_1.htm](http://www.geocities.com/cis261/lab14_1.htm)
  - [http://www.geocities.com/cis261/lab14\\_2.htm](http://www.geocities.com/cis261/lab14_2.htm)
  - [http://www.geocities.com/cis261/lab14\\_3.htm](http://www.geocities.com/cis261/lab14_3.htm)
  - [http://www.geocities.com/cis261/lab14\\_4.htm](http://www.geocities.com/cis261/lab14_4.htm)
  - [http://www.geocities.com/cis261/lab14\\_5.htm](http://www.geocities.com/cis261/lab14_5.htm)

No credit is given to broken link(s).

## Game Programming

### Lecture #15 Multiple Player Games

**Introduction** A multiplayer game is a game which is played by two or more players. The players might be independent opponents, formed into teams or be just a single team pitted against the game.

JavaScript is not an ideal language for multiplayer games because its code is downloaded locally to the player's computer. To make sure each player can see other player's move, the game code should be written with a server-side language (such as PHP or ASP.NET) that collect and process data from each player's computer and present the output on each player's browser.

So you make your move, and your HTML/JavaScript sends form data to your server script. The script records this move and sends back a web page containing updated display. At this point, if the other player refreshes his screen, they will see the same web page as you. Alternatively you can send back a page that automatically refreshes using metarefresh or JavaScript timer.

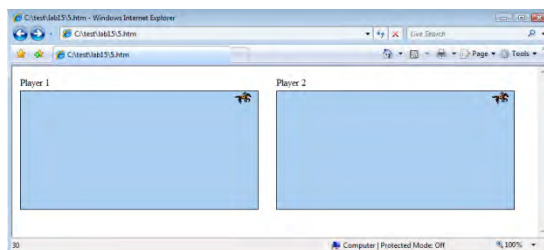
If you just use JavaScript, the results are only visible on your computer - there must be a central point that both of you can look at and the only way is a server web page.

Although this lecture chooses to use PHP and MySQL to explain how a networked game is programmed and played through the Web, PHP and SQL (using MySQL) programming is beyond the scope of this course.

**Standalone multiplayer game** A standalone multiplayer game does not require a computer network. Players can download the game codes and play it locally with multiple sources of user inputs. This lecture discusses two popular modes:

- One player at a time.
- Simultaneous gameplay.

In the simultaneous gameplay mode, two or more players play simultaneously by sending user inputs to the same signal receiver. Consider the following example, which allows two players to race the horseback riding. A screen (which is a browser window) contains two panels, each supports an individual player. Each player presses a particular key once to start the running of the horse.



The coding mechanism is simple.

```
function keyed() {  
  
    if(event.ctrlKey) {  
        if(i==0) { horse1(); i++; }  
    }  
  
    if(event.keyCode==39) {  
        if(j==0) { horse2(); j++; }  
    }  
}
```

```

status = hrs1.style.pixelWidth;
}
.....
<body onLoad="init()" onKeyDown="keyed()">

```

When any of the two players presses the designated key (Shift or the → arrow key), the **keyed()** function will call the **horse1()** or **horse2()** functions accordingly. The **horse1()** function, for example, will move the first horse (with an ID “**hrs1**”) at a speed of either 1 pixel or 2 pixels per 50 milliseconds towards the right. The movement is terminated when **hrs1** hits the right border of the panel it belongs to.

```

function horse1() {
  if ( hrs1.style.pixelLeft + hrs1.style.pixelWidth >=
player1.style.pixelWidth) {
    clearTimeout(s2); // terminate hrs2
    clearTimeout(s1); // terminate hrs1
  }
  else { hrs1.style.pixelLeft += Math.floor(Math.random()*2)+1;
    s1 = setTimeout("horse1()", 50); }
}

```

Notice that there are two **clearTimeout()** methods used in the above code. One is used to terminate **hrs1**, while the other terminates **hrs2**.

In the one-player-at-a-time mode, the programmer’s job is to find a way to count the player’s turns. The programming trick is technically simple. Declare a variable that counts how many times the game has been restarted. In a two-player situation, the number is either an odd or even number. When the number is odd, it is the first player’s turn, when the number is even, it is the second player’s turn.

Consider the following code sample,

```

.....
var player_turn;
.....
function player() {
  player_turn++;
  ball.style.display = "inline"
  ball.style.pixelTop = 0;
  ball.style.pixelLeft = Math.floor(Math.random()*250);
  moveBall();

  if (player_turn%2 == 0) {
    pl.innerText = 2;
    area.style.backgroundColor="yellow";
    ball.style.backgroundColor="blue";
  }
  else {
    pl.innerText = 1;
    area.style.backgroundColor="#abcdef";
    ball.style.backgroundColor="red";
  }

}
.....
<button onClick="player()"
style="position:absolute;top:400">Change Hand</button>
.....

```

In this code, the variable **player\_turn** is incremented by 1 each time when the **player()** function is called by clicking the button. The following statement checks if the current value is an even number.

```
if (player_turn%2 == 0) {
```

If so, the following statements will indicate that it is the second player's turn, and the background color is changed to yellow, while the ball is changed to blue.

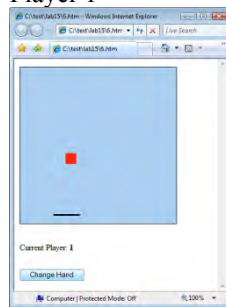
```
p1.innerText = 2;
area.style.backgroundColor="yellow";
ball.style.backgroundColor="blue";
```

If the current number is an odd number, the computer will know it is the first player's turn and carry out the following code segment.

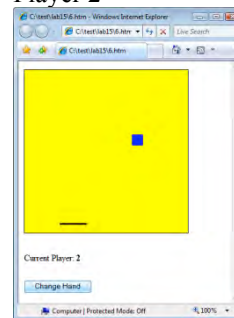
```
p1.innerText = 1;
area.style.backgroundColor="#abcdef";
ball.style.backgroundColor="red";
```

The following outputs will thus display alternatively on the screen.

Player 1



Player 2



Online  
(Networked)  
multiplayer  
game

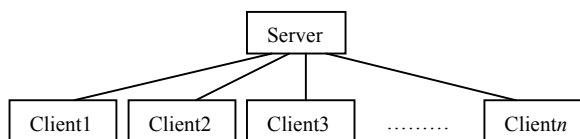
Networked games have existed for the last decade but their underlying software architectures have been poorly designed with little in the way of reusability. A networked game is a software system in which multiple users interact with each other in real-time, even though those users may be physically located around the world.

Typically, each user accesses his/her own computer workstation or console, using it to provide a user interface to the game environment. These environments usually aim to provide users with a sense of realism by incorporating realistic 3D graphics, spatial sound & other modalities to create an immersive experience.

Usually there is a server that synchronizes the game data among users. One can try to understand this model through the analogy of a database model used in the airport. In an airport terminal, crew members work on different workstations to assign seat numbers to airplane passengers. When one crew member confirms and secures a seat for a passenger through a workstation, the workstation will immediately transmits the data to the server. The server will simultaneously notify each every other workstation that a seat is taken, and consequently each seat is assigned to one and the only one passenger.

Networked game typically must integrate with database systems that store persistent information about the game environment. These databases include, for example, detailed information about the environment's terrain elevation, the location of buildings and other static items in the environment, and the initial Networked game configuration.

In a networked multiplayer game, each player makes movements and actively responds to the game themes. There will be many data (each player generates some of them) to be exchanged, processed, and calculated by the centralized control server and each participant computer.



In this model, clients (workstation) must maintain a live (active) connection with the server for bilateral (two-way) communication. Data can then transmit from the client to the server and vice versa simultaneously.

A networked multiplayer game is a game that accepts inputs from multiple sources. Consider the following game code, in which one can use the Shift and Control keys to control the Up and Down direction of an object (e.g. a red square), the other uses  $\uparrow$  and  $\downarrow$  arrow keys to controls the second object (e.g. a blue square). Although the input device is the same--keyboard, there are two sources of user input.

```

<script>

function move() {
    document.body.focus();

    // red
    if (event.shiftKey) { red.style.pixelTop --; }
    else if (event.ctrlKey) { red.style.pixelTop ++; }

    // blue
    switch (event.keyCode) {
        case 38:
            blue.style.pixelTop --; break;

        case 40:
            blue.style.pixelTop ++; break;
    }
}
.....
</script>
  
```

Two players can simultaneously use two set of keys to control two individual objects. However, both players must look at the same screen and use the same keyboard. This is a physical limitation of JavaScript, as a client-side scripting language.

A popular solution to this physical boundary is the so-called AJAX (**A**synchronous **J**avaScript **A**nd **X**ML), which combines the technologies of HTML, CSS, JavaScript, DHTML, and DOM (Document Object Model) with an ability to work with server-side scripting language. The core technology is the implementation of **XMLHttpRequest** object allows web programmers to retrieve data from the web server as a background activity.

When used in game programming, AJAX has four main components. JavaScript defines game rules and program flow. The Document Object Model and Cascading Style Sheets set the appearance in response to data fetched in the background from the server by the XMLHttpRequest object.

Consider the following code segment,

```

<script type="text/javascript">

var xmlhttp=false;

if (!xmlhttp && typeof XMLHttpRequest!='undefined') {
    xmlhttp = new XMLHttpRequest();
}

function ajax_call() {
    xmlhttp.open("GET", 'http://cis261.comyr.com/sqrgame.php?x1=' +
        document.getElementById('red').style.pixelLeft +
        '&y1=' + document.getElementById('red').style.pixelTop +
        '&x2=' + document.getElementById('blue').style.pixelLeft +
        '&y2=' + document.getElementById('blue').style.pixelTop , true);

    xmlhttp.onreadystatechange=function() {
        if (xmlhttp.readyState==4) {
            document.getElementById('bt').innerHTML = xmlhttp.responseText;
        }
    }

    xmlhttp.send(null)
    return false;
}
.....
</script>
.....
<p id="bt"></p>

```

The following is a valid URL that links the instructor's demo site.

<http://cis261.comyr.com/sqrgame.php>

In the above example, when the `ajax_call()` function is called, the client browser will attach four parameters (`x1`, `y1`, `x2`, and `y2`) to the URL, as shown below. Each parameter represents the current coordinate pair of the red and blue square.

<http://cis261.comyr.com/sqrgame.php?x1=21&y1=74&x2=159&y2=367>

The GET method, specified in the `xmlhttp.open()` method delivers these four parameter and their associated values to the web server (which is the host “cis261.comyr.com” in this case), and then particularly to the “sqrgame.php” server-side script. The source code of “sqrgame.php”, written in PHP, looks:

```

<?php

$x1=$_GET["x1"];
$y1=$_GET["y1"];
$x2=$_GET["x2"];
$y2=$_GET["y2"];

$con = mysql_connect("localhost", "MyUserID", "MyPassWord");
if( mysql_select_db("DatabaseName") )
{
    $sql = "UPDATE Coords set x1='$x1', y1='$y1', x2='$x2', y2='$y2' where
    cid='c1'";

    $query = mysql_query($sql);

    $sql2 = mysql_query("select * from Coords where cid='c1'");

    if($sql2) {
        while($row = mysql_fetch_array($sql2)){

```

```

        echo $row['x1'] . ":" . $row['y1'] . ":" . $row['x2'] . ":" . $row['y2'] . ":";
    }
}

mysql_close();

}
else
{
    die( "Error! Could not connect the database: " . mysql_error() );
}
?>

```

The following reads the HTTPREQUEST value sent by the client browser, which is “http://cis261.comyr.com/sqrgame.php?x1=21&y1=74&x2=159&y2=367”

```

$x1=$_GET["x1"];
$y1=$_GET["y1"];
$x2=$_GET["x2"];
$y2=$_GET["y2"];

```

Consequently, 21 is assigned to the PHP variable \$x1, 74 to \$y1, 159 to \$x2, and 367 to \$y2. The following SQL-embedded PHP statement, at the server side, will replace the variables with correct values.

```

UPDATE Coords set x1='$x1', y1='$y1', x2='$x2', y2='$y2'
where cid='c1'

```

In other words, the PHP interpreter converts the above line to the following, and passes the completed SQL statement to MySQL server.

```

UPDATE Coords set x1=21, y1=74, x2=159, y2=367 where cid='c1'

```

The MySQL server then stores these data into a table named “Coords”. A visualization of the table is:

```

mysql> select * from Coords where cid='c1';
+-----+-----+-----+-----+-----+
| cid | x1 | y1 | x2 | y2 |
+-----+-----+-----+-----+
|  c1 | 21 | 74 | 159 | 367 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

```

The following SQL-embedded PHP statements will retrieve the newly added values of x1, y2, x2, and y2, and output them to the receiver. The receiver, in this case, is the **ajax\_call()** method on the client side.

```

$sql2 = mysql_query("select * from Coords where cid='c1'");

if($sql3) {
    while($row = mysql_fetch_array($sql3)){
        echo $row['x1'] . ":" . $row['y1'] . ":" . $row['x2'] . ":" . $row['y2'] . ":";
    }
}

```

The echo command outputs the values to the xmlhttp.**responseText** property (of the ajax\_call() method) in a pattern similar to the following. In other words, the new value of xmlhttp.responseText is:

```

21:74:159:367:

```

The programmer can then write JavaScript code to separate the string value “**21:74:159:367:**” into an array. For example,

```
var str = xmlhttp.responseText;
var crd = str.split(":");
```

The **split()** method is used to split a string into an array of strings. The syntax is:

```
ObjectID.split(separator);
```

After going through the splitting, the string value “**21:74:159:367:**” is broken down to four elements: 21, 74, 159, and 367. The following statement

```
var crd = str.split(":");
```

is equivalent to

```
var crd= new Array(21, 74, 159, 367);
```

To retrieve the values of each element, use **crd[0]**, **crd[1]**, **crd[2]**, and **crd[3]**. Consequently, the **xmlhttp.onreadystatechange** property will, based on the following DHTML statements, assign the initial value of the (x, y) coordinates to the red and blue squares and send them to the client browser, each time when the player access the game.

```
function init() {
.....
xmlhttp.onreadystatechange=function() {
.....
var str = xmlhttp.responseText;
.....
code = "<span id='red' style='background-
Color:red;left:"+crd[0]+";top:"+crd[1]+"'></span>";

code += "<span id='blue' style='background-
Color:blue;left:"+crd[2]+";top:"+crd[3]+"'></span>";

bd.innerHTML = code;
.....
}
}
.....
<body id="bd" onLoad="init()" onKeyDown="move()">
```

To build a new MySQL database with a new table using PHP with embedded SQL statements, use:

```
<?php
$con = mysql_connect("localhost","sqr_game","cis261");
if (!$con)
{
die('Could not connect: ' . mysql_error());
} // Create database
if (mysql_query("CREATE DATABASE sqr_game",$con))
{
echo "Database created";
}
else
{
echo "Error creating database: " . mysql_error();
}
```

```

// Create table
mysql_select_db("a6732376_game", $con);

$sql = "CREATE TABLE Coords
(
cid varchar(15) NOT NULL,
x1 int NOT NULL,
y1 int NOT NULL,
x2 int NOT NULL,
y2 int NOT NULL
)";

// Execute query
mysql_query($sql,$con);

mysql_close($con); // destroy the connection to database
?>

```

When the game is involved in database, the network-based delays can be very significant. These network delays are particularly difficult to handle when multiple users or components interact with each other directly.

For example, a network game must support accurate collision detection, agreement, and resolution among participants. Accurate collision detection is difficult because at any given point in time, no user has accurate information about the other users' current positions; as we have seen, the network delay means that all received information is out-of-date.

It is possible, therefore, that one user might conclude, based on stale information, which a collision occurred, while, in fact, the other user actually moved to avoid the collision during the network delay period.

## Game Programming

### Lab #15

### Multiple Player Games

#### Learning Activity #1: A simple two-player game

1. Use Notepad to create a new text file named lab15\_1.htm, with the following contents:

```
<html>

<style>
  div { height:200; border:solid 1 black; background-color:#abcdef; position:
absolute;}
  img { position:relative; }
</style>

<script>
var i=0; j=0;

function init() {
player1.style.pixelWidth = 400;
player2.style.pixelWidth = 400;
hrs1.style.pixelWidth = 30;
hrs2.style.pixelWidth = 30;
}

function keyed() {

  if(event.ctrlKey) {
    if(i==0) { horse1(); i++; }
  }

  if(event.keyCode==39) {
    if(j==0) { horse2(); j++; }
  }

status = hrs1.style.pixelWidth;
}

function horse1() {

  if ( hrs1.style.pixelLeft + hrs1.style.pixelWidth >= player1.style.pixelWidth) {
    clearTimeout(s2);
    clearTimeout(s1);
  }
  else { hrs1.style.pixelLeft += Math.floor(Math.random()*2)+1;
s1 = setTimeout("horse1()", 50); }
}

function horse2() {

  if ( hrs2.style.pixelLeft + hrs2.style.pixelWidth >= player2.style.pixelWidth) {
    clearTimeout(s1);
    clearTimeout(s2);
  }
  else { hrs2.style.pixelLeft += Math.floor(Math.random()*2)+1;
s2 = setTimeout("horse2()", 50); }
}

</script>
```

```

<body onLoad="init()" onKeyDown="keyed()">

<table border=0 width="100%">

<tr>
  <td>Player 1</td><td>Player 2</td>
</tr>

<tr>
  <td width="50%"><div id="player1">
    

  </div></td>

  <td width="50%"><div id="player2">
    

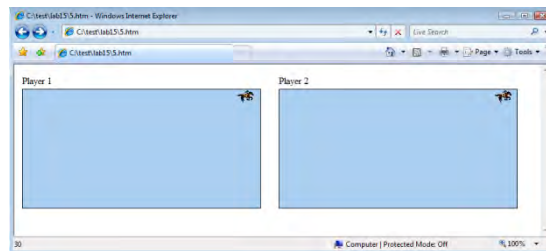
  </div></td>
</tr>

</table>

</body>
</html>

```

2. Test the program with Internet Explorer. Player 1 uses the “Ctrl” key and player 2 use the → arrow key to start the game simultaneously. A sample output looks:



## Learning Activity #2: Two-player game – one player at a time

1. Use Notepad to create a new text file named lab15\_2.htm, with the following contents:

```

<html>

<style>
div { position:absolute; width:300; height:300; border:solid 1 black; left:10; }
span { position:relative; width:20; height:20; background-Color: red}
hr { position:absolute; top:280}
</style>

<script>
var code="";

var player_turn;

function init() {
  var i = Math.floor(Math.random()*250);
  var j = Math.floor(Math.random()*280);

  code = "<span id='ball' style='left:"+j+"'></span>";
  code += "<hr id='bar' size='3' width='50' color='black' style='left:"+i+"'>";
  area.innerHTML = code;

```

```

    area.style.backgroundColor="#abcdef";

    player_turn = 1;
    pl.innerText = player_turn;
    moveBall();
}

function moveBall() {
    if (ball.style.pixelTop >= 275) {
        clearTimeout(s1);
        ball.style.display = "none"
    }

    else if ((ball.style.pixelTop + ball.style.pixelHeight >= 260) &&
        (ball.style.pixelLeft >= bar.style.pixelLeft) &&
        (ball.style.pixelLeft + 20 <= bar.style.pixelLeft + 50))
    {
        ball.style.pixelTop = 0;
        ball.style.pixelLeft = Math.floor(Math.random()*250);
    }

    else { ball.style.pixelTop += 2; }
    s1 = setTimeout("moveBall()", 20);
}

function player() {
    player_turn++;
    ball.style.display = "inline"
    ball.style.pixelTop = 0;
    ball.style.pixelLeft = Math.floor(Math.random()*250);
    moveBall();

    if (player_turn%2 == 0) {
        pl.innerText = 2;
        area.style.backgroundColor="yellow";
        ball.style.backgroundColor="blue";
    }
    else {
        pl.innerText = 1;
        area.style.backgroundColor="#abcdef";
        ball.style.backgroundColor="red";
    }
}

function moveBar() {
    switch(event.keyCode) {
        case 37:
            if (bar.style.pixelLeft<=10) { bar.style.pixelLeft = 10; }
            else { bar.style.pixelLeft-=2; }
            break;

        case 39:
            if (bar.style.pixelLeft >= 250) { bar.style.pixelLeft = 250; }
            else { bar.style.pixelLeft+=2; }
            break;
    }
}
</script>

<body onload="init()" onKeyDown="moveBar()">

```

```

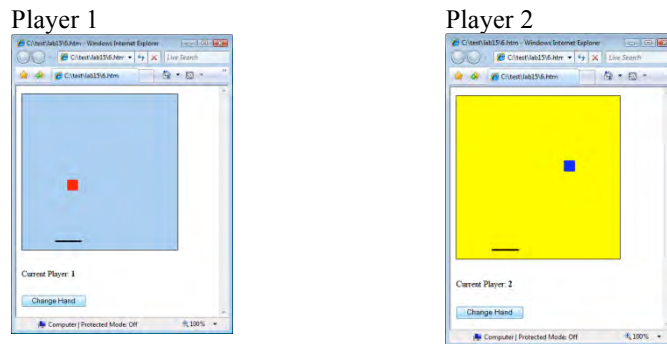
<div id="area">

</div>

<p style="position:absolute;top:350">Current Player: <b id="p1"></b></p>
<button onClick="player()" style="position:absolute;top:400">Change Hand</button>
</body>
</html>

```

2. Test the program with Internet Explorer. A sample output looks:



### Learning Activity #3:

1. Use Notepad to create a new text file named lab15\_3.htm, with the following contents:

```

<html>
<style>
span { position:absolute; width:20px; height:20px; }
</style>

<script>

function move() {
    document.body.focus();

// red
if (event.shiftKey) { red.style.pixelTop --; }
else if (event.ctrlKey) { red.style.pixelTop ++; }

// blue
switch (event.keyCode) {
case 38:
    blue.style.pixelTop --; break;

case 40:
    blue.style.pixelTop ++; break;
}
}

function init() {
x1 = Math.floor(Math.random()*90) + 10;
y1 = Math.floor(Math.random()*140) + 10;

x2 = Math.floor(Math.random()*90) + 10;
y2 = Math.floor(Math.random()*140) + 10;

x3 = Math.floor(Math.random()*90) + 10;
y3 = Math.floor(Math.random()*140) + 10;

```

```

    red.style.pixelLeft = x1;
    red.style.pixelTop = y1;

    blue.style.pixelLeft = x2;
    blue.style.pixelTop = y2;

    green.style.pixelLeft = x3;
    green.style.pixelTop = y3;

    action();
}

var i=1;
var j=1;
var k=2;

function action() {

    if (red.style.pixelTop <=10 || red.style.pixelTop >=150) {
        i = i * -1;
    }

    if (blue.style.pixelTop <=10 || blue.style.pixelTop >=150) {
        j = j * -1;
    }

    if (green.style.pixelLeft <=10 || green.style.pixelLeft >=100) {
        k = k * -1;
    }

    red.style.pixelTop += i;
    blue.style.pixelTop += j;
    green.style.pixelLeft += k;

    setTimeout("action()", 50);
}

</script>

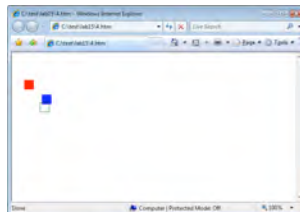
<body onLoad="init()" onKeyDown="move();">

<span id="red" style="background-color:red;"></span>
<span id="blue" style="background-color:blue;"></span>

<span id="green" style="border: solid 1 green;"></span>
</body>
</html>

```

2. Test the program with Internet Explorer. A sample output looks:



#### Learning Activity #4: Networked Two-Player Game

1. Use Notepad to create a new text file named lab15\_4.htm, with the following contents:

```

<html>

<style>
span {position:absolute; width:20; height:20}
</style>

<script type="text/javascript">

//var req=false;
var xmlhttp=false;

if (!xmlhttp && typeof XMLHttpRequest!='undefined') {
    xmlhttp = new XMLHttpRequest();
}

function ajax_call() {
    xmlhttp.open("GET", 'http://cis261.comyr.com/sqrgame.php?x1=' +
        document.getElementById('red').style.pixelLeft +
        '&y1=' + document.getElementById('red').style.pixelTop +
        '&x2=' + document.getElementById('blue').style.pixelLeft +
        '&y2=' + document.getElementById('blue').style.pixelTop , true);

    xmlhttp.setRequestHeader("Content-Type", "text/html");

    xmlhttp.onreadystatechange=function() {
        if (xmlhttp.readyState==4) {
            status = xmlhttp.responseText;
            var str = xmlhttp.responseText;
            var crd = str.split(":");

            status = "red:("+crd[0]+", "+crd[1]+") blue:("+crd[2]+", "+crd[3]+")";

        }
    }

    xmlhttp.send(null)
    return false;

    setTimeout("ajax_call()",100);
}

function init() {
    xmlhttp.open("GET", "http://cis261.comyr.com/sqrinit.php" , true);

    xmlhttp.setRequestHeader("Content-Type", "text/html");

    xmlhttp.onreadystatechange=function() {
        if (xmlhttp.readyState==4) {
            //status = xmlhttp.responseText;
            var str = xmlhttp.responseText;
            var crd = str.split(":");
            status = "red:("+crd[0]+", "+crd[1]+") blue:("+crd[2]+", "+crd[3]+")";
            var code="";

            code = "<span id='red' style='background-
Color:red;left:"+crd[0]+";top:"+crd[1]+" '></span>";
            code += "<span id='blue' style='background-
Color:blue;left:"+crd[2]+";top:"+crd[3]+" '></span>";
            bd.innerHTML = code;
        }
    }
}

```

```

xmlhttp.send(null)
return false;
}

function move() {

if(event.shiftKey) {
switch(event.keyCode) {
case 37:
red.style.pixelLeft --; break;

case 39:
red.style.pixelLeft ++; break;

case 38:
red.style.pixelTop --; break;

case 40:
red.style.pixelTop ++; break;
}
}

if(event.ctrlKey) {
switch(event.keyCode) {
case 37:
blue.style.pixelLeft --; break;

case 39:
blue.style.pixelLeft ++; break;

case 38:
blue.style.pixelTop --; break;

case 40:
blue.style.pixelTop ++; break;
}
}

ajax_call();
}

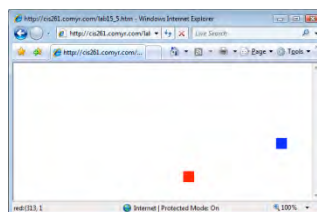
</script>

<body id="bd" onLoad="init()" onKeyDown="move()">

</body>
</html>

```

2. Test the program with Internet Explorer. Hold the Shift key and press →, ↓, ↑, or ← arrow keys to move the red square. Hold the Ctrl key and press →, ↓, ↑, or ← arrow keys to move the blue square. A sample output looks:



### Learning Activity #5:

1. Use Notepad to create a new text file named lab15\_5.htm, with the following contents:

```
<html>

<style>
  span { background-Color: white; border:solid 1 black;
  width:50; height:50; text-align:center;
  }
</style>

<script>
var req;

var code = "";
var mymark;

var cell = new Array();

function reset() {
  for (k=1; k<=9; k++) {
    eval("c"+k+".innerText=' '");
  }
  save_mark();
}

function init() {

  for (i=1; i<=9; i++) {
    code += "<span id='c'+i+' ' onClick='set_mark()'></span>";
    if (i%3==0) { code += "<br>"; }
  }
  area.innerHTML = code;

  init_request() ;
}

function init_request() {
  if (window.XMLHttpRequest) {
    req = new XMLHttpRequest();
  }
  else if (window.ActiveXObject) {
    req = new ActiveXObject("Microsoft.XMLHTTP");
  }

  req.open('GET', "tictoe.php", true);

  req.setRequestHeader("Content-Type", "text/html");

  req.onreadystatechange = processResponse;

  req.send(null);
}

function player_mark() {
  frm.innerText = "You chose : " + mymark;
}

function set_mark() {
  var keyID = event.srcElement.id;
  document.getElementById(keyID).innerHTML = "<b>" + mymark + "</b>";
}
```

```

    cell[keyID] = mymark;
    save_mark();
}

function save_mark() {

    if (window.XMLHttpRequest) {
        req = new XMLHttpRequest();
    }
    else if (window.ActiveXObject) {
        req = new ActiveXObject("Microsoft.XMLHTTP");
    }

    var url = "tictoe.php" +
        "?c1=" + cell['c1'] +
        "&c2=" + cell['c2'] +
        "&c3=" + cell['c3'] +
        "&c4=" + cell['c4'] +
        "&c5=" + cell['c5'] +
        "&c6=" + cell['c6'] +
        "&c7=" + cell['c7'] +
        "&c8=" + cell['c8'] +
        "&c9=" + cell['c9'];

    req.open('GET', url, true);

    req.setRequestHeader("Content-Type", "text/html");

    req.onreadystatechange = processResponse;

    req.send(null);
}

function processResponse() {
    if(req.readyState == 4) {

        var str = req.responseText;

        status = str;
        var crd = str.split(":");

        c1.innnerText = crd[0];
        c2.innnerText = crd[1];
        c3.innnerText = crd[2];
        c4.innnerText = crd[3];
        c5.innnerText = crd[4];
        c6.innnerText = crd[5];
        c7.innnerText = crd[6];
        c8.innnerText = crd[7];
        c9.innnerText = crd[8];

        var crd = str.split(":");

    }
    //setTimeout("save_mark()",100);
}

</script>

<body onLoad="init()">

<form id="frm">Choose your mark:

```

```

<input type="radio" name="mark" onClick="mymark='O'; player_mark()">O
<input type="radio" name="mark" onClick="mymark='X'; player_mark()">X
</form>

<div id="area">

</div>

<p><button onClick="reset()">Reset</button></p>

</body>

</html>

```

2. Test the program with Internet Explorer. A sample output looks:



## Game Programming

### Lecture #14 Using Yahoo! User Interface (YUI) Library

**Introduction** The Yahoo! User Interface (YUI) Library is a set of utilities and controls, written in JavaScript, for building richly interactive web applications using techniques such as DOM (Document Object Model) scripting, DHTML and AJAX. The YUI Library also includes several core CSS resources. All components in the YUI Library have been released as open source under a BSD license and are free for all uses.

**The YUI Event Utility** The YUI Event Utility is designed for creating event-driven applications. Yahoo says it can make in-browser programming easier because there is no full-featured JavaScript or CSS library that can work in every browser. Yahoo believes their YUI can support the vast majority of browsers, and thus make your web programming journey easier.

All the events provided by DOM (Document Object Model) are in the yahoo-dom-event.js library file. DOM is an application programming interface (API). It defines the logical structure of documents and the way a document is accessed and manipulated. With the DOM programmers can build documents, navigate their structure, and add, modify, or delete elements and content.

Technically speaking, the YUI Event Utility is actually a set of script files. Yahoo currently allows programmers to externally access this file and includes it in HTML files. Simply add the following bold-faced lines next to the <HTML> tag.

```
<html>
<script type="text/javascript"
src="http://yui.yahooapis.com/2.5.2/build/yahoo-dom-
event/yahoo-dom-event.js"></script>
```

To illustrate event handling syntax, the instructor creates an area using <div> and </div> tags.

```
<div id="btn"> Click Me</div>
```

The following CSS style code defines the appearance of this area.

```
<style type="text/css">
#btn {background-color:#abcdef; border: solid 1 black;
width:100; cursor:pointer; text-align:center}
</style>
```

Next, create a function that receives a single argument, the event object (e), and pops up an alert which says "Welcome to CIS262!":

```
var msg = function(e) {
    alert("Welcome to CIS262!");
}
```

This area (with and ID btn) must be associated with the function that process the event. You need to use the Event Utility's **addListener** method to tie the **msg** function with the **btn** area, so it will serves as a handler for the **click** event. Add the following line:

```
YAHOO.util.Event.addListener("btn", "click", msg);
```

The complete source code now looks:

```
<html>
```

```

<script type="text/javascript"
src="http://yui.yahooapis.com/2.5.2/build/yahoo-dom-
event/yahoo-dom-event.js"></script>

<script>
(function() {
var msg = function(e) {
    alert("Welcome to CIS262!");
}

YAHOO.util.Event.addListener("btn", "click", msg);

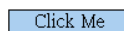
})();
</script>

<style type="text/css">
#btn {background-color:#abcdef; border: solid 1 black;
width:100; cursor:pointer; text-align:center}
</style>

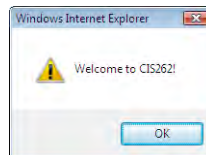
<body>
<div id="btn"> Click Me</div>
</body>
</html>

```

When testing the program, you first see the following:



Clicking on it, The Yahoo Event utility fires a pop-up window that looks:



Notice that the **YAHOO.util.Event.on** is an alias for `addListener`. In the above example, you can use the following line instead.

```
YAHOO.util.Event.on("btn", "click", msg);
```

A complete list of event utility is available at <http://developer.yahoo.com/yui/docs/YAHOO.util.Event.html>.

#### Using the YUI Dom Collection

The Dom Collection comprises a family of convenience methods that simplify common DOM-scripting tasks, including element positioning and CSS style management, while normalizing for cross-browser inconsistencies. It contains these subsections:

- Positioning HTML Elements
- Getting and Setting Styles
- Getting the Viewport Size
- Managing Class Names

Consider the following example. It uses the following two methods of the YUI Dom Collection

- **getXY**: get an element's position relative to the document.
- **setXY**: position an element relative to the document.

```
<html>
```

```

<script type="text/javascript"
src="http://yui.yahooapis.com/2.5.2/build/yahoo-dom-event/yahoo-
dom-event.js"></script>

<script>
(function() {
  var move = function(e) {
    YAHOO.util.Dom.setXY('square', YAHOO.util.Event.getXY(e));
  };

  YAHOO.util.Event.on(document, "click", move);

})();

</script>

<style type="text/css">
#square {width:20px; height:20px;background-color:#00f;}
</style>

<span id="square"></span>
</html>

```

The following line uses the **getXY()** method to retrieve the *x*- and *y*-coordinates of the mouse cursor when the user click the mouse relative to the HTML document in the browser. It then uses the **setXY()** method to force the “square” object to move to that position from wherever it is.

```
YAHOO.util.Dom.setXY('square', YAHOO.util.Event.getXY(e));
```

#### Using the Drag & Drop Utility

The term “**drag and drop**” is defined as moving an element from one location to another by dragging it with the mouse. It is a technique to directly manipulate an object by moving it and placing it somewhere else using mouse.

The **Drag & Drop Utility** allows you to create a draggable interface efficiently, buffering you from browser-level abnormalities and enabling you to focus on the interesting logic surrounding your particular implementation. This component enables you to create a variety of standard draggable objects with just a few lines of code and then, using its extensive API, add your own specific implementation logic.

This Drag & Drop Utility is developed based on the following logic:

```

The basics are straightforward: the user holds down the mouse
button while the mouse hovers over a page element, and the user
then moves the mouse with the button still depressed. The
element follows the mouse around until the user finally
releases it.

```

Consider the following object and its styles. It is a red square that has width and height of 50 pixels and a black border. Its id is “red”.

```

<style type="text/css">
span { width:50; height:50; cursor:default; border:solid 1
black}
</style>
.....
<span id="red" style="background-Color:red"></span>

```

You need to include the following library file in your web page with the script tag:

```
<script type="text/javascript">
```

```
src="http://yui.yahooapis.com/2.5.2/build/yahoo-dom-
event/yahoo-dom-event.js"></script>
```

To use Drag and Drop utilities, include the following library file in your web page with the script tag:

```
<script type="text/javascript"
src="http://yui.yahooapis.com/2.5.2/build/dragdrop/dragdrop-
min.js"></script>
```

To enable a simple drag and drop of any DOM element, create a new instance of **YAHOO.util.DD**, providing the constructor with the HTML id of the element you want to make draggable. Optionally, you can provide an element reference instead of an id. For example,

```
var r = new YAHOO.util.DD("red");
```

Alternatively you can use an element reference. For example,

```
var r1=YAHOO.util.Dom.get("red");
var r = new YAHOO.util.DD(r1);
```

In this example, a new instance of **YAHOO.util.DD** is created for an element whose HTML id is "red"; this enables "red" to be dragged and dropped.

When dragging an object, the **onDOMReady** event handler is fired automatically. You can then use it create a visual effect of moving the object. A sample way to create such function is:

```
<script type="text/javascript">
(function() {
  var r;
  YAHOO.util.Event.onDOMReady(function() {
    r = new YAHOO.util.DD("red");
  });
})();
</script>
```

Note that while the element can now be moved visually on the page, this visual motion is accomplished by changing the element's style properties. Its actual location in the DOM itself remains unchanged.

The basic approach used for drag-and-drop is as follows:

- Event handlers inspect the incoming event to determine the element being dragged.
- A **mousedown** handler saves the starting co-ordinates, sets the zIndex so that the element appears in front during the drag, changes some other style settings to indicate a drag has begun, and performs other initialisation.
- A **mousemove** handler inspects the mouse's co-ordinates and moves the element accordingly using its left and top properties. Here's where cross-browser support gets nasty - mouse co-ordinates in the event object are seriously platform-specific.
- A **mouseup** handler restores normal style settings and performs any other cleaning up.

Using the  
Animation Utility

The Animation Utility enables the rapid prototyping and implementation of animations involving size, opacity, color, position, and other visual characteristics. **YAHOO.util.Anim** is the base animation class that provides the interface for building animated effects. Its syntax is:

```
YAHOO.util.Anim ( el , attributes , duration , method );
```

Its parameters are:

- *el* <String | HTML<Element> Reference to the element that will be animated

- *attributes* <Object> The attribute(s) to be animated. Each attribute is an object with at minimum a "to" or "by" member defined. Additional optional members are "from" (defaults to current value), "units" (defaults to "px"). All attribute names use camelCase.
- *duration* <Number> (optional, defaults to 1 second) Length of animation (frames or seconds), defaults to time-based
- *method* <Function> (optional, defaults to YAHOO.util.Easing.easeNone) Computes the values that are applied to the attributes per frame (generally a YAHOO.util.Easing method)

To use the Animation utility, include the following source file:

```
<script
src="http://yui.yahooapis.com/2.5.2/build/animation/animation-
min.js" type="text/javascript"></script>
```

Consider the following example. In it, the instructor creates an instance of **YAHOO.util.Anim** to animate an object.

```
<html>

<script type="text/javascript"
src="http://yui.yahooapis.com/2.5.2/build/yahoo-dom-
event/yahoo-dom-event.js"></script>
<script type="text/javascript"
src="http://yui.yahooapis.com/2.5.2/build/animation/animation-
min.js"></script>

<button id="btn">Vanish</button>
<div id="bar" style="background-color:red; width:70%"></div>

<script type="text/javascript">
(function() {
  var attributes = {
    width: { to: 0 }
  };

  var anim = new YAHOO.util.Anim('bar', attributes);

  YAHOO.util.Event.on('btn', 'click', function() {
    anim.animate();
  });

})();
</script>

</html>
```

The object to be animated is a red bar defined by:

```
<div id="bar" style="background-color:red; width:100%"></div>
```

A button is used to fire the event defined in `YAHOO.util.Event.on` method:

```
<button id="btn">Vanish</button>
```

The final step is to call the `animate` method on our instance to start the animation. The button will be the trigger that begins the animation sequence:

```
YAHOO.util.Event.on('btn', 'click', function() {
  anim.animate();
});
```

```
});
```

When the “Vanish” button is clicked, the width of the red bar decrease at a constant speed till it disappears.



The YUI Animation Utility includes an Easing feature, which allows you to customize how the animation behaves. For example, add the duration and easing arguments are optional.

```
var anim = new YAHOO.util.Anim('bar', attributes, 2,
YAHOO.util.Easing.easeOut);
```

In this case, the value 2 is the duration of animation. The **easeOut** property will slow the animation gradually as it nears the end.

Similarly, you can change **YAHOO.util.Easing.easeOut** to any of the following for different visual effects:

- YAHOO.util.Easing.backBoth
- YAHOO.util.Easing.backIn
- YAHOO.util.Easing.backOut
- YAHOO.util.Easing.bounceBoth
- YAHOO.util.Easing.bounceIn
- YAHOO.util.Easing.bounceOut
- YAHOO.util.Easing.easeBoth
- YAHOO.util.Easing.easeBothStrong
- YAHOO.util.Easing.easeIn
- YAHOO.util.Easing.easeInStrong
- YAHOO.util.Easing.easeNone
- YAHOO.util.Easing.easeOut
- YAHOO.util.Easing.easeOutStrong
- YAHOO.util.Easing.elasticBoth
- YAHOO.util.Easing.elasticIn
- YAHOO.util.Easing.elasticOut

The Animation Utility also allows you to animate the motion of an **HTMLElement** along a curved path using control points.

Try create a demo object and a button that will move the object:

```
<button id="btn">Move</button> <br/>
<span id="square"></span>
```

Use CSS style to define how this object will look:

```
<style type="text/css">
#square {
background-color:red; height:20px; width:20px;
}
</style>
```

Next, create an instance of **YAHOO.util.Motion**, passing it the element we wish to animate, and the points attribute (an array of [x, y] positions), with the point we are animating to, and

the control points that will influence the path:

```
<script type="text/javascript">
var attributes = {
  points: { to: [10,50], control: [[150,400], [300,100],
[50,800], [450,10] ] }
};

var anim = new YAHOO.util.Motion('square', attributes);
</script>
```

[10,50] is the finish point, namely it is last point the red square will move to. The array, [150,400], [300,100], [50,800], and [450,10], defines the path of the movement. In each set [x,y], the value represents x- and y-coordinates on the HTML document.

The syntax of **YAHOO.util.Motion** is:

```
YAHOO.util.Motion ( el , attributes , duration , method );
```

The parameters are:

- *el* <String | HTMLElement> Reference to the element that will be animated
- *attributes* <Object> The attribute(s) to be animated. Each attribute is an object with at minimum a "to" or "by" member defined. Additional optional members are "from" (defaults to current value), "units" (defaults to "px"). All attribute names use camelCase.
- *duration* <Number> (optional, defaults to 1 second) Length of animation (frames or seconds), defaults to time-based
- *method* <Function> (optional, defaults to YAHOO.util.Easing.easeNone) Computes the values that are applied to the attributes per frame (generally a YAHOO.util.Easing method)

The final step is to call the `animate` method on our instance to start the animation. The button will be the trigger that begins the animation sequence.

```
<script type="text/javascript">
YAHOO.util.Event.on('btn', 'click', function() {
  anim.animate();
});
</script>
```

The complete code will look:

```
<html>
<script type="text/javascript"
src="http://yui.yahooapis.com/2.5.2/build/yahoo-dom-
event/yahoo-dom-event.js"></script>
<script type="text/javascript"
src="http://yui.yahooapis.com/2.5.2/build/animation/animation-
min.js"></script>

<style type="text/css">
#square {
  background-color:red; height:20px; width:20px; left:10;
top:50}
}
</style>

<button id="btn">Move</button><br/>
<span id="square"></span>

<script type="text/javascript">
(function() {
```

```

var attributes = {
  points: { to: [10,50], control: [[150,400], [300,100],
[50,800], [450,10] ] }
};

var anim = new YAHOO.util.Motion('square', attributes);

YAHOO.util.Event.on('btn', 'click', function() {
  anim.animate();
});

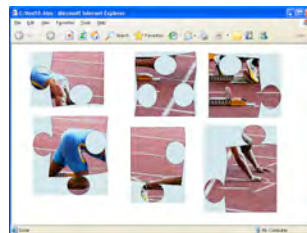
})();
</script>

</html>

```

Coping with the  
IE transparent  
issue

The Portable Network Graphic (PNG) image format was developed by the World Wide Web Consortium as a better GIF than GIF many years ago. It is the trend in the Web developing industry to use .png instead of .gif graphics. However, Internet Explorer in Windows XP (or early) platform supports PNG transparency only up to the 256-color palette. When the PNG transparent image uses a better palette, IE will make transparent parts opaque. For example,



Web developers have long complained about this problems. Microsoft admits the problem (<http://support.microsoft.com/?scid=kb;en-us;265221>) and offers a workaround that's not entirely satisfactory.

Microsoft's AlphaImageLoader uses DirectX components to produce the same transparency effect (<http://msdn.microsoft.com/workshop/author/filter/reference/filters/AlphaImageLoader.asp> ). It's a filter you can apply to any other HTML element, even a block of text. Here it's applied to DIV.

```

<DIV ID="oDiv" STYLE="position:absolute; left:140px; height:400; width:400;
filter:progid:DXImageTransform.Microsoft.AlphaImageLoader (src='image.png', sizing
Method='scale');" >
</DIV>

```

This may not seem that painful, but if you have several PNGs on your page, you need to repeat this for each one. Obviously this workaround is a lot more complicated to use than a simple <img> tag.

Review Questions

- Which statement is true about the Yahoo! User Interface?
  - It is a set of utilities and controls, written in C++, for building application.
  - It is written in Java using techniques such as CSS and DHTML.
  - It is a set of toolkit Yahoo! developed to sell for profits.
  - None of the above.

D

- Which statement is not true?

- A. The YUI Event Utility is designed for creating event-driven applications.
- B. DOM is short for Data Object Mode.
- C. There is no full-featured JavaScript or CSS library that can work in every browser.
- D. All the events provided by DOM are in the yahoo-dom-event.js library file

B

3. Given the following code segment, which statement is correct?

```
var msg = function(e) {
    alert("Welcome to CIS262!");
}
```

- A. var is the name of function.
- B. msg is the name of function.
- C. msg is a variable that holds the value "Welcome to CIS262!".
- D. alert is a variable that holds the value "Welcome to CIS262!".

B

4. Which has exactly the same functionality as the following:

```
YAHOO.util.Event.addListener("btn", "click", msg);
```

- A. YAHOO.util.Event.on("btn", "click", msg);
- B. YAHOO.util.Event.addListener("btn", "click", msg);
- C. YAHOO.util.Event.loadListener("btn", "click", msg);
- D. YAHOO.util.Event.get("btn", "click", msg);

A

5. Which method of the YAHOO.util.Dom class can position an element relative to the document?

- A. setXY
- B. getXY
- C. readXY
- D. writeXY

A

6. Given the following code segment, which statement is correct?

```
YAHOO.util.Dom.setXY('square', YAHOO.util.Event.getXY(e));
```

- A. It uses the getXY() method to retrieve the x- and y-coordinates of the object e.
- B. It uses the setXY() method to retrieve the x- and y-coordinates of the object e.
- C. It uses the getXY() method to force the "square" object to move to a new position from wherever it is.
- D. All of the above.

A

7. The term "drag and drop" is defined as \_\_\_\_.

- A. clicking a button to change its location.
- B. clicking an icon to change its location.
- C. moving an element from one location to another by dragging it with the mouse.

D. moving an element from one location to another by holding the Shift key.

C

8. Which class is the one that support "drag and drop"?

- A. YAHOO.util.Dom
- B. YAHOO.util.Event
- C. YAHOO.util.DD
- D. YAHOO.util.Anim

C

9. Given the following code segment, which is the name of an animated object?

```
var anim = new YAHOO.util.Anim('bar', attributes, 2, YAHOO.util.Easing.easeOut);
```

- A. anim
- B. bar
- C. 2
- D. easeOut

B

10. Given the following code segment, which is the finish point of the movement?

```
points: { to: [10,50], control: [[150,400], [300,100], [50,800]] }
```

- A. (10, 50)
- B. (150, 400)
- C. (300, 100)
- D. (50, 800)

A

## Game Programming

### Lab #14 Using Yahoo! User Interface (YUI) Library

#### Learning Activity #1: Using YUI Event Utility

1. Use Notepad to create a new file named **lab14\_1.htm** with the following contents:

```
<html>
<script type="text/javascript"
src="http://yui.yahooapis.com/2.5.2/build/yahoo-dom-event/yahoo-dom-event.js">
</script>

<script>
(function() {
  var msg = function(e) {
    alert("Welcome to CIS262!");
  }

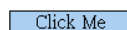
  YAHOO.util.Event.addListener("btn", "click", msg);

})();
</script>

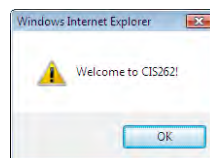
<style type="text/css">
#btn {background-color:#abcdef; border: solid 1 black; width:100; cursor:pointer;
text-align:center}
</style>

<body>
<div id="btn"> Click Me</div>
</body>
</html>
```

2. Use Internet Explorer to test the code. Testing the program, and click on the following button.



3. Clicking on it, The Yahoo Event utility fires a pop-up window that looks:



#### Learning Activity #2: Using Drag-&-Drop Library

1. Use Notepad to create a new file named **lab14\_2.htm** with the following contents:

```
<html>
<head>

<script type="text/javascript"
src="http://yui.yahooapis.com/2.5.2/build/yahoo-dom-event/yahoo-dom-event.js">
</script>

<script type="text/javascript"
src="http://yui.yahooapis.com/2.5.2/build/dragdrop/dragdrop-min.js">
</script>

<style type="text/css">
```

```

span { width:50; height:50; cursor:default; border:solid 1 black}
</style>

<script type="text/javascript">
(function() {
  var r, g, b;
  YAHOO.util.Event.onDOMReady(function() {
    r = new YAHOO.util.DD("red");
    g = new YAHOO.util.DD("green");
    b = new YAHOO.util.DD("blue");
  });
})();
</script>

</head>

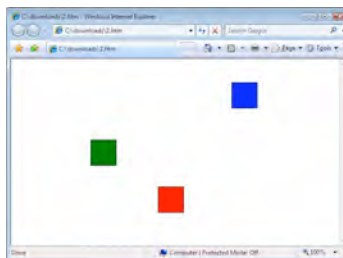
<body>

<span id="red" style="background-Color:red"></span>
<span id="green" style="background-Color:green"></span>
<span id="blue" style="background-Color:blue"></span>

</body>
</html>

```

2. Use Internet Explorer to test the code.



### Learning Activity #3: Animation with different visual effects

1. Use Notepad to create a new file named **lab14\_3.htm** with the following contents:

```

<html>

<script type="text/javascript"
src="http://yui.yahooapis.com/2.5.2/build/yahoo-dom-event/yahoo-dom-event.js">
</script>
<script type="text/javascript"
src="http://yui.yahooapis.com/2.5.2/build/animation/animation-min.js"></script>

<button id="btn">Vanish</button>
<button id="btn_e">Vanish with easeOut</button>
<button id="btn_b">Vanish with backIn</button><p>

<div id="red" style="background-Color:red; width:90%"></div><br />
<div id="blue" style="background-Color:blue; width:90%"></div><br />
<div id="yellow" style="background-Color:yellow; width:90%"></div><br />

<script type="text/javascript">
(function() {
  var attributes = {
    width: { to: 0 }
  };
};

```

```

var anim = new YAHOO.util.Anim('red', attributes);
var anim_e = new YAHOO.util.Anim('blue', attributes, 2,
YAHOO.util.Easing.easeOut);
var anim_b = new YAHOO.util.Anim('yellow', attributes, 2,
YAHOO.util.Easing.backIn);

YAHOO.util.Event.on('btn', 'click', function() {
    anim.animate();
});

YAHOO.util.Event.on('btn_e', 'click', function() {
    anim_e.animate();
});

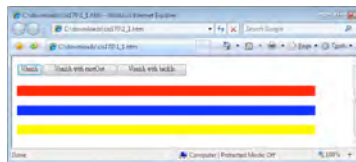
YAHOO.util.Event.on('btn_b', 'click', function() {
    anim_b.animate();
});

})();
</script>

</html>

```

2. Use Internet Explorer to test the code. Click the buttons to see the different visual effects.



#### Learning Activity #4: An online puzzle game

Note: If you are using a browser that supports .png transparency, you can use the shortened code in Appendix A.

1. Use Notepad to create a new file named **lab14\_4.htm** with the following contents:

```

<html>
<head>

<script type="text/javascript"
src="http://yui.yahooapis.com/2.5.2/build/yahoo-dom-event/yahoo-dom-event.js">
</script>

<script type="text/javascript"
src="http://yui.yahooapis.com/2.5.2/build/dragdrop/dragdrop-min.js">
</script>

<style type="text/css">
img { cursor:default; position:absolute}
</style>

<script type="text/javascript">
function init() {
m1.style.pixelLeft = Math.floor(Math.random()*(document.body.clientWidth-200));
m1.style.pixelTop = Math.floor(Math.random()*(document.body.clientHeight-200));
m2.style.pixelLeft = Math.floor(Math.random()*(document.body.clientWidth-200));
m2.style.pixelTop = Math.floor(Math.random()*(document.body.clientHeight-200));
m3.style.pixelLeft = Math.floor(Math.random()*(document.body.clientWidth-200));
m3.style.pixelTop = Math.floor(Math.random()*(document.body.clientHeight-200));
m4.style.pixelLeft = Math.floor(Math.random()*(document.body.clientWidth-200));

```

```

m4.style.pixelTop = Math.floor(Math.random()*(document.body.clientHeight-200));
m5.style.pixelLeft = Math.floor(Math.random()*(document.body.clientWidth-200));
m5.style.pixelTop = Math.floor(Math.random()*(document.body.clientHeight-200));
m6.style.pixelLeft = Math.floor(Math.random()*(document.body.clientWidth-200));
m6.style.pixelTop = Math.floor(Math.random()*(document.body.clientHeight-200));
}

(function() {

    YAHOO.util.Event.onDOMReady(function() {
        var m1 = new YAHOO.util.DD("m1");
        var m2 = new YAHOO.util.DD("m2");
        var m3 = new YAHOO.util.DD("m3");
        var m4 = new YAHOO.util.DD("m4");
        var m5 = new YAHOO.util.DD("m5");
        var m6 = new YAHOO.util.DD("m6");

    });
})();
</script>

</head>

<body onLoad="init()">
<DIV ID="m1" STYLE="position:absolute; width:128; height:97;
    filter:progid:DXImageTransform.Microsoft.AlphaImageLoader(
        src='http://business.cypresscollege.edu/~pwu/cis262/1.png',
        sizingMethod='scale');">
</DIV>

<DIV ID="m2" STYLE="position:absolute; width:127; height:153;
    filter:progid:DXImageTransform.Microsoft.AlphaImageLoader(
        src='http://business.cypresscollege.edu/~pwu/cis262/2.png',
        sizingMethod='scale');">
</DIV>

<DIV ID="m3" STYLE="position:absolute; width:126; height:113;
    filter:progid:DXImageTransform.Microsoft.AlphaImageLoader(
        src='http://business.cypresscollege.edu/~pwu/cis262/3.png',
        sizingMethod='scale');">
</DIV>

<DIV ID="m4" STYLE="position:absolute; width:95; height:132;
    filter:progid:DXImageTransform.Microsoft.AlphaImageLoader(
        src='http://business.cypresscollege.edu/~pwu/cis262/4.png',
        sizingMethod='scale');">
</DIV>

<DIV ID="m5" STYLE="position:absolute; width:130; height:138;
    filter:progid:DXImageTransform.Microsoft.AlphaImageLoader(
        src='http://business.cypresscollege.edu/~pwu/cis262/5.png',
        sizingMethod='scale');">
</DIV>

<DIV ID="m6" STYLE="position:absolute; width:109; height:113;
    filter:progid:DXImageTransform.Microsoft.AlphaImageLoader(
        src='http://business.cypresscollege.edu/~pwu/cis262/6.png',
        sizingMethod='scale');">
</DIV>

</body>
</html>

```

2. Use Internet Explorer to test the code.

### Learning Activity #5: Moving along a curve

1. Use Notepad to create a new file named **lab14\_5.htm** with the following contents:

```
<html>
<script type="text/javascript" src="http://yui.yahooapis.com/2.5.2/build/yahoo-dom-
event/yahoo-dom-event.js"></script>
<script type="text/javascript"
src="http://yui.yahooapis.com/2.5.2/build/animation/animation-min.js"></script>

<style type="text/css">
#square {
  background-color:red; height:20px; width:20px; left:10; top:50;
}
</style>

<button id="btn">Move</button><br/>
<span id="square"></span>

<script type="text/javascript">
(function() {
  var attributes = {
    points: { to: [10,50], control: [[150,400], [300,100], [50,800], [450,10] ] }
  };

  var anim = new YAHOO.util.Motion('square', attributes);

  YAHOO.util.Event.on('btn', 'click', function() {
    anim.animate();
  });

})();
</script>

</html>
```

2. Use Internet Explorer to test the code. Click the button to move the object along a curve.

### Submittal

1. Complete all the 5 learning activities in this lab.
2. Create a .zip file named lab4.zip containing ONLY the following self-executable files.
  - Lab14\_1.htm
  - Lab14\_2.htm
  - Lab14\_3.htm
  - Lab14\_4.htm
  - Lab14\_5.htm
3. Log in to Blackboard, and enter the course site.
4. Upload the zipped file to question 11 of Assignment 04 as response.

## Appendix A:

```
<html>
<head>

<script type="text/javascript"
  src="http://yui.yahooapis.com/2.5.2/build/yahoo-dom-event/yahoo-dom-event.js">
</script>

<script type="text/javascript"
  src="http://yui.yahooapis.com/2.5.2/build/dragdrop/dragdrop-min.js">
</script>

<style type="text/css">
img { cursor:default; position:absolute}
</style>

<script type="text/javascript">
function init() {
m1.style.pixelLeft = Math.floor(Math.random()*(document.body.clientWidth-200));
m1.style.pixelTop = Math.floor(Math.random()*(document.body.clientHeight-200));
m2.style.pixelLeft = Math.floor(Math.random()*(document.body.clientWidth-200));
m2.style.pixelTop = Math.floor(Math.random()*(document.body.clientHeight-200));
m3.style.pixelLeft = Math.floor(Math.random()*(document.body.clientWidth-200));
m3.style.pixelTop = Math.floor(Math.random()*(document.body.clientHeight-200));
m4.style.pixelLeft = Math.floor(Math.random()*(document.body.clientWidth-200));
m4.style.pixelTop = Math.floor(Math.random()*(document.body.clientHeight-200));
m5.style.pixelLeft = Math.floor(Math.random()*(document.body.clientWidth-200));
m5.style.pixelTop = Math.floor(Math.random()*(document.body.clientHeight-200));
m6.style.pixelLeft = Math.floor(Math.random()*(document.body.clientWidth-200));
m6.style.pixelTop = Math.floor(Math.random()*(document.body.clientHeight-200));
}

(function() {

  YAHOO.util.Event.onDOMReady(function() {
    var m1 = new YAHOO.util.DD("m1");
    var m2 = new YAHOO.util.DD("m2");
    var m3 = new YAHOO.util.DD("m3");
    var m4 = new YAHOO.util.DD("m4");
    var m5 = new YAHOO.util.DD("m5");
    var m6 = new YAHOO.util.DD("m6");

    });
  })();
</script>

</head>

<body onload="init()">








</body>
</html>
```

## Game Programming

### Lecture #14 Basics of 2D/3D Graphics

**Using graphics software** In computer graphics, graphics software or image editing software is a program or collection of programs that enable a person to manipulate visual images on a computer. They are very useful in producing high-quality graphics and image files. Many games use such pre-produced graphics and images to enhance their multimedia performance.

Most graphics programs have the ability to import and export one or more graphics file formats. Several graphics programs support animation, or digital video. Vector graphics animation can be described as a series of mathematical transformations that are applied in sequence to one or more shapes in a scene. Raster graphics animation works in a similar fashion to film-based animation, where a series of still images produces the illusion of continuous movement.

**Graphics and Images types** Graphics images created for use in the World Wide Web have the file name extensions: **.gif** (Graphics Interchange Format), **.jpg** or **jpeg** (pronounced as jaypeg) or **.png** (stands for portable network graphics and is pronounced as 'ping'). There are many more less popular graphics formats used on the world wide web. You can discover their formats by viewing their file extensions. The file extension is the three letters that come after the file name.

The .gif, .jpg (jpeg) and .png file formats allow image data to be stored in a compressed form that maximizes the amount of space used but describes the colors, dimensions, and excepting jpg format, any transparent background in the image.

The gif and png formats permit transparent backgrounds, making them suitable for logos (like the cockerel), web buttons etc. When you don't want to have a colored background around the image, like this, but want the background page color to show through. Gifs are used for animated graphics. Both gif and png are best used when the image contains only a few colors and are excellent for line drawings. PNG format graphics may not be visible in all browsers.

The jpg (jpeg) format is good for images that contain a range of colors and shades, e.g. photographs but isn't much good for any images with sharp edges, including lettering or line drawings.

The png format is expected to eventually replace gif. It supports transparency and is better than gif at displaying color, although image sizes tend to be larger.

A general rule is that the images have to be of a fairly small size otherwise they take too long to load when people are looking at web pages.

**Animated Gifs** The so-called "**Animated GIF**" is an animation created by combining multiple GIF images in one file. The result is multiple images, displayed one after another, that give the appearance of movement. You can use a GIF animation tool to create sequences of images to simulate animation and allows for transparent background colors. Animated GIFs can generate higher response rates than static banners.

**Creating Animated Gifs** You can start with a check list of all the things you need to create animated gifs.

- An imaging software such as paint shop pro or color works,
- A gif assembling software such as Gif animator, Animation Shop, Giffy,
- Creativity, and
- A lot of patience

A number of software programs are available either for free or shareware on the Internet. We










recommend Paint Shop Pro (JASC. Inc.) as the imaging software, and Gif Animator (Ulead). You may be familiar with other software programs. A complete list of software programs available can be found [here](#).

Creating animated gifs is really simple. Let us start off with an example. Below is an image of a ball.



The aim of this first exercise is to make the ball move from left to right and then back. In the imaging software often a new work area 450 pixels wide and height equal to the original image (in this case it is 49 pixels). Copy the ball and paste it into the working area at the far left.

Create a new working area same height and width. Paste the ball at a position more at the right than the previous frame. Repeat this procedure until the ball is completely at the right. At this point you should have a number of frames like these shown below.

File 1	
File 2	
File 3	
File 4	
File 5	
File 6	
File 7	
File 8	
File 9	

Now import the individual images in the gif assembler program in sequential order, and then in the reverse order. Save the animation. The end result should look something like this:



You can use Internet Explorer to visit

[http://business.cypresscollege.edu/~pwu/cis261/files/rolling\\_ball.gif](http://business.cypresscollege.edu/~pwu/cis261/files/rolling_ball.gif) to see how this gif file is animated.

## Transparency

A major problem which you might encounter when creating animated gifs is that the background of the image does not blend so well with the background you are using. This can be solved in two ways:

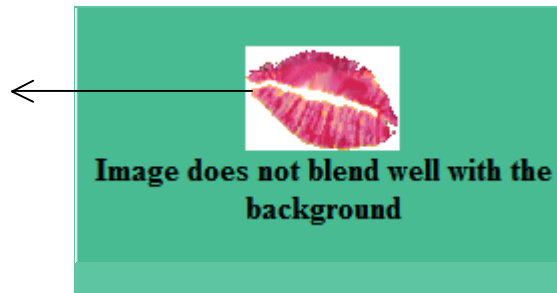
- Change the background color of the animation to the one in your webpage, or else
- Create a transparent background

The use of transparent backgrounds allows us to blend any kind of animation (or still gifs) to any background being used. Consider the following image.

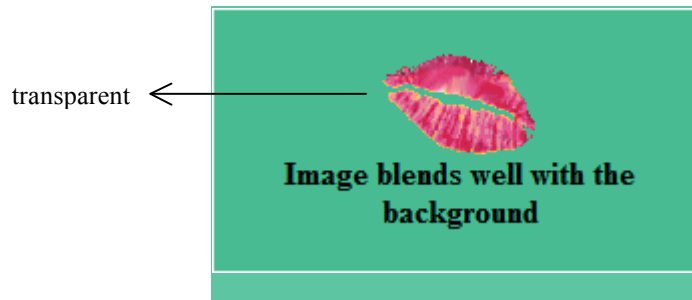


This image blends well with the white background which this page has. The reason is that the background color of the animation is also white.

Now change the background color. Notice that the image will not blend so well as above.



Now change the white background to a transparent one. The result is the following.



Transparent backgrounds may seem the best way how to blend animations to the background. At times however the end result will not be the expected one. Let us consider the following image:



As you can clearly see the different frames of the animation remained visible. This is a common problem when using transparencies on moving objects. This can be easily solved. One of the options of every gif animation program is the way how the image (frame) will be removed from the screen. Usually the current options are available (notice that the name of option varies from software to software):

- Do Not Remove
- To Background Color
- To Previous Image

In the above example the animation was set to Do Not Remove. On changing the settings so that each frame is removed **To background Color**, so the results looks:

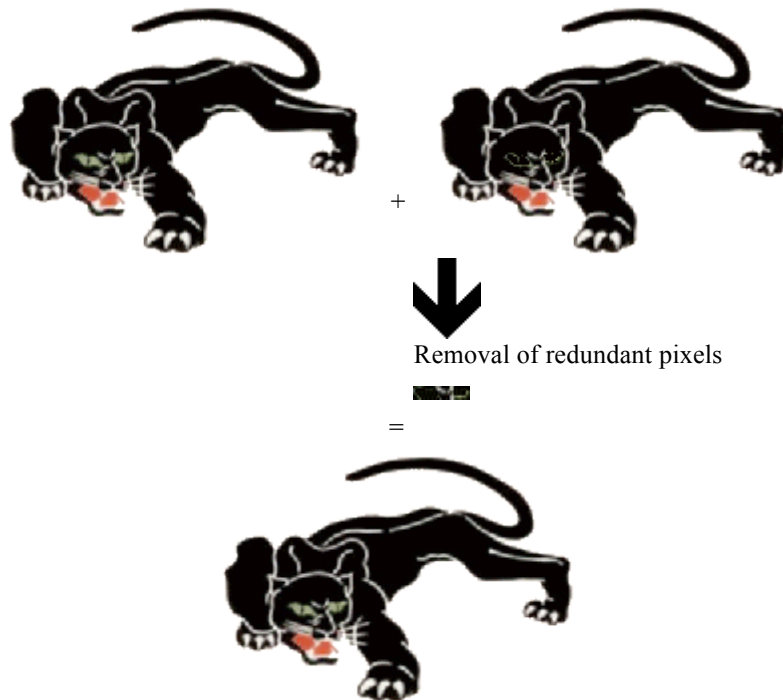


Reducing the file size

File size is a very important thing which needs to be kept in mind when creating animated gifs. The smaller they are in size the faster they will load on the web page. It is not the first time that animations produced have a file size over 20K. This will mean that they will take far longer to download.

Well software producers have done something to help us by creating the so called gif reducers. What is the function of these programs? These programs will act in 3 different ways:

- **Palette reducers** - This feature will reduce the number of colors within the animation from the original number (e.g. 256) to a user defined or predefined number (e.g. 64). Obviously the image will sometimes loose in quality and become dithered. In this case one can change the number of resulting colors to an acceptable level.
- **Removal of Redundant pixels** - This is a very cool feature. Basically the software will compare the different frames and will remove/crop any pixels which do not change from one image to another. Consider the following panther. It is composed of 2 frames. In the initial frame the panther has the eyes open. In the second frame the tiger has the eyes closed. The software cropped off the pixels which do not change.



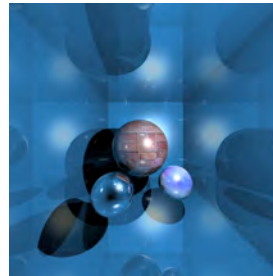
This process has reduced the file size by 5,279 bytes which is equivalent to 63% of the original file size.

- **Removal of Comment Blocks** - Comment blocks are text annotations added to the images which do not appear in the animation. These annotations usually state who created the animated gif, and which program was used in creating them. Although the amount of bytes added to the file size may be few, some programs remove these annotations as well.

These features are shared in the majority of software packages available.

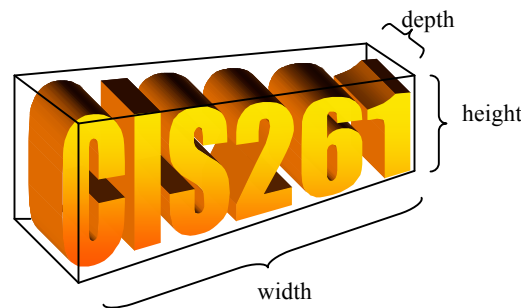
## 3D graphics

3D computer graphics are works of graphic art created with the aid of digital computers and 3D software. The term may also refer to the process of creating such graphics, or the field of study of computer graphic techniques and related technology. An example of 3D graphics is:



Each 3D object takes up space in three directions, defined in 3D parlance as the **X**, **Y**, and **Z** axes. Many computer languages refer the three axes are **width**, **height**, and **depth**.

- **width**: the object's left-to-right dimension
- **height**: the object's top-to-bottom dimension
- **depth**: the object's front-to-back dimension



Take a close look at the above illustration, which the instructor created for this class. 3D computer graphics are different from 2D computer graphics in that a three-dimensional representation of geometric data is stored in the computer for the purposes of performing calculations and rendering 2D images. Such images may be for later display or for real-time viewing.

3D modeling is the process of preparing geometric data for 3D computer graphics, and is akin to sculpting, whereas the art of 2D graphics is analogous to photography. Despite these differences, 3D computer graphics rely on many of the same algorithms as 2D computer graphics.

In computer graphics software, the distinction between 2D and 3D is occasionally blurred; 2D applications may use 3D techniques to achieve effects such as lighting, and primarily 3D may use 2D rendering techniques.

Although this class is not designed for you to learn 3D graphics, do not hesitate to download some 3D modeling software and try for yourself. Use Internet Explorer to visit <http://www.freerifsoftware.com/software/3dPlus/default.asp>, and download a copy of 3DPlus 2 (which is a completely FREE 3D animation and modeling software).

By the way, currently, OpenGL and Direct3D are two popular APIs for generation of real-time imagery. Real-time means that image generation occurs in 'real time', or 'on the fly', and may be highly user-interactive. Many modern graphics cards provide some degree of

hardware acceleration based on these APIs, frequently enabling display of complex 3D graphics in real-time.

Review Questions

8. Which graphics formats permit transparent backgrounds?

- A. .jpg
- B. .wma
- C. .png
- D. .bmp

9. To allow an image to blends wells with the background of a given area, you need to make sure the image file is \_\_\_\_.

- A. intangible
- B. impermeable
- C. transparent
- D. invisible

10. Which is not a common dimension of a 3D object?

- A. width
- B. height
- C. depth
- D. breadth

## Game Programming

Lab #15

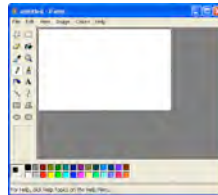
Game Graphics

### Preparation #1:

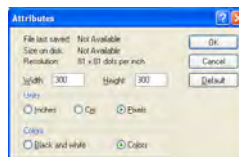
3. Create a new directory named **C:\games**.
4. Use Internet Explorer to visit <http://business.cypresscollege.edu/~pwu/cis261/files/lab2.zip> and download the zipped file. Extract the files to C:\games directory. Make sure the C:\games directory contains:
  - star1.gif
  - star2.gif
  - logo1.gif
  - logo2.gif
  - logo3.gif
5. Use Internet Explorer to visit <http://www.freerissoftware.com/software/PhotoPlus/>, and download a copy of PhotoPlus 6 (which is a completely **FREE** image and photo editing software) to your C:\ drive.
6. Install the PhotoPlus 6 software. An installation tip is available at [http://www.freerissoftware.com/software/PhotoPlus/getting\\_started.asp](http://www.freerissoftware.com/software/PhotoPlus/getting_started.asp).

### Learning Activity #1: Using Microsoft Paint to create 2D image file

1. Click Start, Programs, Accessories, and then Paint to open the Microsoft Paint.



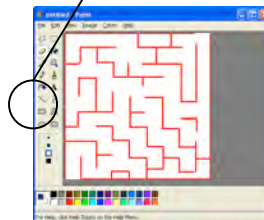
2. On the toolbar, click Image, and then Attributes..
3. Set the image width to 300 pixels and height to 300 pixels, as shown below.



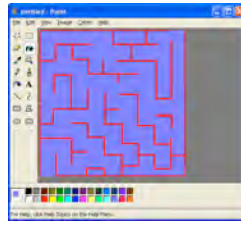
4. Click OK.
5. On the toolbox, click the Line icon.



6. Change the color to red.
7. Create a labyrinth. For example,



8. Add a background color if you wish. For example,



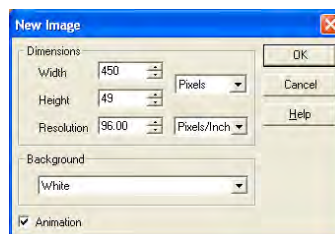
9. Save the file as **C:\games\lab2\_1.gif** for later use. (You will use this image file in a later lecture.)

### Learning Activity #2: Creating animated Gif file (using PhotoPlus 6.0 or your preferred tool)

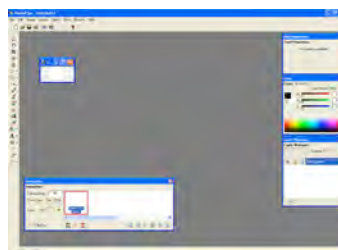
1. Click Start, (All) Programs, and then Serif PhotoPlus6 to launch the program.



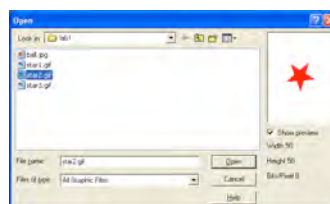
2. Click the Create New Animation option.
3. Set the Width to be **50**, and Height to be **50**. Click OK.



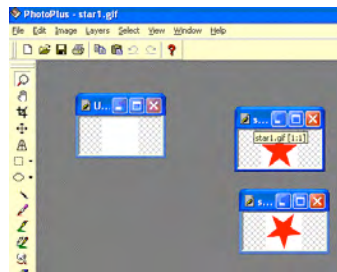
4. The screen now looks:



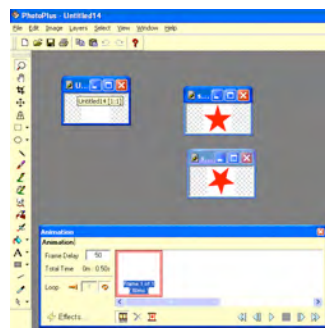
5. Click File, and Open, and then locate the star1.gif file to open it.



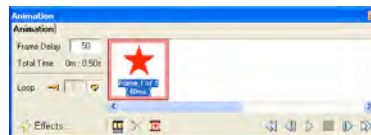
6. Open star2.gif, too.
7. Click on **star1.gif**, click Edit, and then Copy.



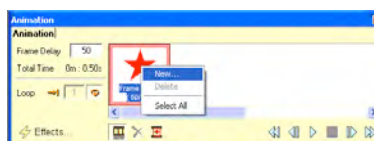
8. Click the **Untitled $n$**  (where  $n$  is a number), click Edit, Paste, and then As New Layer.



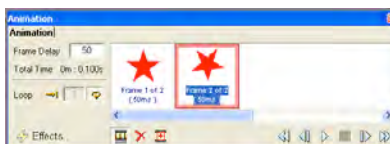
9. The Animation Tab now looks:




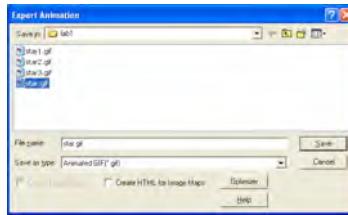
10. Right click on the **Frame 1 of 1** picture, as shown below, and then click New...



11. Click on **star2.gif**, click Edit, and then Copy.
12. Click the **Untitled $n$**  (where  $n$  is a number), click Edit, Paste, and then As New Layer.
13. The Animation Tab now looks:



14. Click the  button to test the animation. The star should now rotate.
15. To export the artwork, click File, and the Export....



16. Save the file as **C:\games\lab2\_2.gif**.

### **Misc. functions**

typeof null;

num.toString(radix);  
parseInt("3 blind mice");  
parseInt("0xFF");  
parseInt("ff", 16);  
parseInt("eleven");  
javascript:alert("Hi there");  
eval("3 + 12");  
escape(str);

unescape(str);

Returns string "object" (or whatever the thing is).  
Returns num var as a string var.  
Returns 3.  
Returns 255.  
Returns 255 also.  
Returns NaN.  
Execute JS as a URL.  
Evaluates a text string as code.  
Returns str in web-compliant form. E.g.  
"hi ho" = "hi%20ho".  
Returns str from web-compliant form.  
Undoes escape().

### **Window functions**

alert("Are you sure about that?");  
confirm("Continue loading page?");

prompt("Enter your name, please.");  
close();  
find(), home(), print(), stop();  
focus(), blur();

moveBy(), moveTo();  
resizeBy(), resizeTo();  
scrollBy(), scrollTo();

var intervalID = setInterval("bounce()", 10000);

clearInterval(intervalID);

setTimeout("display\_time()", 10000);

clearTimeout();

var w = window.open("smallwin.html", "SmallWin",  
"width=400,height=300,status,resizeable,menubar");

Prompts with OK box.  
Prompts with OK / CANCEL message box.  
Prompts with text box.  
Close the window.  
Duplicates of buttons. Not in IE4.  
Cause us focus, or lose focus. Not in IE3.  
Move the window.  
Resize the window.  
Scroll the document displayed in the window.  
Set func to be repeatedly called w/ delay. Call OUT of func.  
Cancel function to be repeatedly invoked with delay.  
Call display\_time() in 1 sec. Put in display\_time() to loop.  
Cancel function invoked *once* after delay.  
Opens another window, loading smallwin.html, with name SmallWin, and given dimensions.

### **Window properties**

closed  
(useful for open()).  
defaultStatus, status  
line (in status bar).  
document  
(the HTML page).  
frames[]  
history  
represents browsing hist.  
history.back();  
history.forward();  
history.go();  
IE3 – best avoid.  
innerHeight, innerWidth, outerHeight, outerWidth  
window; not in IE4.  
locationbar, menubar, personalbar, scrollbars, statusbar, toolbar  
IE4.  
name  
TARGET>, for ex.  
opener  
null if opened by user.

Returns true if window has been closed  
  
Sets default status line, current status  
  
Refers to the current document object  
  
Refers to frames, if any.  
A reference to the history object,  
  
Go back a link.  
Go forward a link.  
Goes to a link, buggy in NS2&3, weird in  
  
Inner and outer dimensions of the  
  
References to visibility of parts. Not in  
  
Name of current window. Useful for <A  
  
Reference to Window opened this, or

### **Document functions**

document.write("<h2>Table of Factorials</h2>");  
document.writeln("Hi there.");  
document.forms[i].elements[j]++;

Outputs to current doc, writes HTML.  
Outputs with a <CR> appended at end.  
Access forms and form elements via array scripting.

`document.close();`

Closes this window-if opened by JS, in later browsers.

### **Document properties**

`document.location`

Set to load new doc.

`document.forms[0]`, `document.myform`

`document.alinkColor`

Represents URL document displayed.

Refer to document forms.

Color of hyperlink while clicked on (same as `<BODY>` tag).

Hyperlink array.

Applet array.

Background color of document.

Allows JS to read / write cookies. `== ""` if not set.

Embedded array.

Text color of document (same as `<BODY>` tag).

Images array.

Returns string of the date we were last modified.

Color of unclicked links. Same as LINK attr. in `<BODY>`.

Links array.

URL of doc that ref'd us, if any.

The title (`<TITLE>`) of this document.

URL we were loaded from, same as `location.href`.

Visited link color. Same as VLINK in `<BODY>`.

Returns the name of the domain you're currently at.

`document.anchors[]`

`document.applets[]`

`document.bgColor = "#040404";`

`document.cookie`

`document.embeds[]`

`document.fgColor = "blue";`

`document.images[]`

`document.lastModified`

`document.linkColor`

`document.links[]`

`document.referrer`

`document.title`

`document.URL`

`document.vlinkColor`

`document.domain`

### Navigator properties

navigator.appName  
navigator.appVersion

navigator.userAgent

navigator.appCodeName

navigator.platform  
navigator.language

navigator.userLanguage, navigator.systemLanguage

navigator.javaEnabled()

The simple name of the web browser.  
The version number and/or other version info about browser.  
appName and appVersion combined, usually.  
The code name of the browser. E.g., "Mozilla."  
Platform they're running on as of JS1.2.  
Language of browser. "en" (English). NS4+, not IE.  
IE4+ version of navigator.language property.  
Returns true if Java supported and enabled on this browser.

### Math functions

Math.round(x/15);  
Math.pow(x,y);  
Math.sqrt(x\*x + y\*y);  
Math.random();  
Math.max(i, j);  
Math.min(i, j);  
Math.floor(j);  
Math.ceil(j);

Rounds to the nearest integer.  
Returns  $x^y$ .  
Returns square root of argument.  
Returns random between 0.0 – 0.1.  
Returns greater of two numbers.  
Returns lesser of two numbers.  
Rounds j down.  
Rounds j up.

### String functions

str.length;  
str.charAt(str.length - 1);

str.substring(1, 4);  
str.indexOf('a'), str.lastIndexOf(" ");

str.anchor(name), str.big(), str.blink(), str.bold(), str.fixed(), str.italics(), str.link(href);

str.small(), str.strike(), str.sub(), str.sup()

str.fontcolor("#090909"), str.fontSize(1-7 | "+2");  
str.match(), str.replace(), str.search();

str.slice(2[, 6]);

str = "1,2,3,4,5"; arr = str.split(",");

str.substr(5,2);  
str.toUpperCase(), str.toLowerCase();

Returns a string's character length.  
Returns the last character of a string.  
SEE NOTES BELOW.  
Returns str[1] through str[3].  
Returns position of first "a" / last " " in string str, -1 if none.  
Return str with certain formatting imposed upon it.  
anchor=<A NAME=name>, link=<A HREF>, rest obvious.  
Set the string's font color / size.  
Regexp / string match / replace functions.  
Returns str[2] through str[5], neg. args start from end.  
Returns array of substrings, split by delimiter ",".  
Returns str[5] through str[5+2].  
Convert a string's case.

### Vital notes

- • **Semi-colons** are optional, but **recommended**.
- • JavaScript is **case-sensitive**; HTML embedded names (such as onClick) are **not**.
- • Always declare variables with **var**. Variables not declared with **var** are global automatically. Keep vars declared on top for clarity. Ex: var ind = 0;
- • Fun with strings: **"Hi there"** and **'Hi there'** are both legal string definitions.
- • **Octal** number definitions begin with a 0. **Hex** begins with 0x (or 0X). Ex: 026, 0xAF, 0377, 0xff...
- • JavaScript represents all numbers as floating point. **Numbers can be extremely large**, like: -999 tril <-> +999 tril.
- • **String indexing**, like str[str.length - 1], is supported by Nav4+, not IE4 though (IE5?).
- • **null** is a special value in JS. It is **not** equivalent to 0. It represents the lack of an object, number, string, etc. Sometimes, converted to 0, though.
- • **Functions** can be **nested** since JS1.2.
- • **undefined** can be tested for by making an uninitialized variable: var undefined; if (myform["checkbox" + ind] == undefined) ...

### Useful code tidbits

```
<input type = "button" ... onClick = "alert('You clicked me!')">
```

```
var square = new Function("x", "return x*x;");
definition.
var square = function(x) { return x*x; }
execute it.
image.width <-> image["width"]
var pattern = new RegExp("\bjava\b", "i");
expression).
var o = new Object();
up properties.
var point = { x:2.3, y:-1.2 };
properties.
var sq = { upleft: { x:point.x, y:point.y }, lowright: { x:(point.x+side), y:(point.y+side) }};
for example.
document.images[i].width;
document object.
var a = new Array(); a[0] = 1.2; a[1] = "JavaScript"; a[2] = true; a[3] = { x:1, y:3 };
elems. added easy.
var a = [1.2, "JavaScript", true, { x:1, y:3 }];
var matrix = [[1,2,3], [4,5,6], [7,8,9]];
var sparseArray = [1,,5];
elements.
for (var i in obj);
properties of an object.
Circle.prototype.pi = 3.14159;
<body bgcolor = "&{favorite_color()};">
embed JS in HTML.
```

### **Notable constants**

Number.MAX\_VALUE  
Number.MIN\_VALUE  
Number.NaN  
Number.POSITIVE\_INFINITY  
Number.NEGATIVE\_INFINITY  
infinity.

When user clicks button, execute  
"onClick" portion.  
Function literal – variable holds function

Function literal. square(144) would

Two ways to access object properties.  
Creates RegExp object (regular-

Makes a general object... you can make

Object literal – general object with init'd

Object literal, with sq.upleft.x == point.x,

Way to access images as array of

Creates an array. Once made, indexed

Alternate way since JS1.2.

Nested array definition.

Makes array with some undefined

The for/in loop loops through the

Sets a pi val for all Circle objects.  
&{ JS-statements; }; used in NS3+, IIE4,

Largest representable number.

Most negative representable number.

Special Not-a-number value.

Special value to represent infinity

Special value to represent negative

## Object-based browser detection

Document object	Browser that supports it
document.images	NS3+, IE4+
!document.images	NS2, IE3
document.layers	NS4+
document.all	IE4+
document.layers    document.all	NS4+, IE4+

if (document.images) document.images[0].src = "/images/myimg1.jpg";  
detection.

Example usage of browser object

## Useful events

Handler	Triggered when	Supported by
OnAbort	Loading interrupted.	Image
OnBlur	Element loses input focus.	Text elms., Window, all other elms.
OnChange	User changes an elm., moves on.	Select, text input elements
OnClick	User single-click. Ret. false = cancel.	Link, button elements
OnError	Error occurs while loading an image.	Image
OnFocus	Element given input focus.	Text elms., Window, all other elms.
OnLoad	Document or image finishes loading.	Window, Image
OnMouseOut	Mouse moves off element.	Link
OnMouseOver	Mouse moves over elm.	Link
OnReset	Form reset request, false = no reset.	Form
OnSubmit	Form submit, false = no submit.	Form
OnUnload	Document is unloaded.	Window

## Cookie stuff

```
document.cookie = "version=" + escape(document.lastModified) +  
converts to web form,  
"; path=/; expires=" + mydate.toGMTString();  
javascript:alert(document.cookie)  
cookie set for site!  
document.cookie = "Name=Joe Bob; path=/";
```

Sets persistent cookie. escape()

unescape() undoes from web form.  
Type this in your browser to see the

Minimal cookie setting.

## Variable arguments

```
function add_all_together() {  
arguments  
    for (i = 0; i < add_all_together.arguments.length; i++)  
arguments  
        total += add_all_together.arguments[i];  
demonstrate  
}
```

The **arguments** method stores the  
themselves, and the number of  
passed to each function, as we  
here.

## Pre-load and update images

```
myimg1 = new Image();  
myimg1.src = "/images/image01.jpg";  
  
function imgfilter(imgobj, newimg) {  
    if (document.images)  
document.images method,  
        document.images[imgobj] = newimg.src;  
object argument.  
}
```

Make a new image object,  
set the image src to preload it.

If the browser supports the  
set a new src for the imgobj image

## Date stuff

```
var now = new Date();  
time.  
var xmas = new Date(97, 11, 25);  
index from 0!  
now.toLocaleString();  
xmas.toGMTString();  
time.
```

Date obj representing current date and

Date obj for 25-Dec-97, note months

Returns string of date and time.  
Returns string of date and time in GMT

## Creating a Plain-Text Document

```
var w = window.open("", "console", "width=600,height=300,resizeable");  
because there's
```

We specify the optional [window.]open

```
w.document.open("text/plain");
w.document.writeln(msg);
```

a document.open() function too.

### **To insure a child window you want to update is still open**

```
if (!w.closed) w.close();
by the user.
```

This will close the window declared above if it's still open, only if that window hasn't been closed

### **To generate a random number between X and Y**

```
function generate( lbound, ubound ) {
    return Math.floor( ( ubound - lbound + 1 ) * Math.random() + lbound );
}
```

// This one I got somewhere. It doesn't work. It returns only odd numbers, and goes 1 beyond the set range sometimes???

```
function generate( x, y ) {
    var range = y-x+1;
    var i = ( "" + range ).length;
    var num = ( Math.floor( Math.random() * Math.pow( 10, i ) ) % range ) +
    parseInt( x );
    return num;
}
```

### **Detecting Shockwave**

```
<SCRIPT LANGUAGE="JavaScript">
<!--hiding contents of script from old browsers, just in case
```

```
//If this browser understands the mimeType property and recognizes the MIME Type //"application/futuresplash"...
if (navigator.mimeType && navigator.mimeType["application/x-shockwave-flash"]){
```

```
    //...write out the following <EMBED> tag into the document.
    document.write('<EMBED SRC="flash_movie.swf" WIDTH="220" HEIGHT="110" LOOP="true" QUALITY="high">');
}
```

```
//Otherwise,...
else {
```

```
    //...write out the following <IMG> tag into the document. The image need
    //not be the same size as the Flash movie, but it may help you lay out the
    //page if you can predict the size of the object reliably.
    document.write('<IMG SRC="welcome.gif" WIDTH="220" HEIGHT="110" ALT="Non-Shockwave Welcome">');
}
```

```
//Done hiding from old browsers. -->
</SCRIPT>
```

### **Using JS to Write to Frames**

Just give the frames names, then access them by name:

```
<frameset cols="*,*">
<frame name=left src="a.html">
<frame name=right src="b.html">
</frameset>
```

```
top . right . document . open ();
top . right . document . writeln ("Hello.");
top . right . document . close ();
```

```
top.document.location = "newpage.htm";
```

Breaks from frames, goes to new page.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	D	D	D	C	B	C						D		
2	B	B	B	D	B	A						B		
3	B	B	C	C	B	D						D		
4	C	B	A	A	A	B						B		
5	B	C	B	A	B	A						A		
6	A	A	A	D	A	A						B		
7	A	A	D	B	C	A						C		
8	C		A	D	D	D						D	C	
9	D		B	B	A	B						D	C	
10	D		C	A	A	A						A	D	

