IST400/600 Scripting for Games, Spring 2010
Lab 6
Instructor: Keisuke Inoue

Description:

In this lab, you will create the **Mini Golf** game that was discussed in the previous class. The following instructions will help you to create the game, but will not tell you all the small steps. The lab is designed to encourage your learning and enhance your understanding by working independently. **You may ask questions to instructors and/or friends (but keep it quiet) and look up previous lecture materials, textbook, and other resources. Do not copy-and-paste any resources other than you created.**

Peer Survey:

The peer survey is back! You will be in one of the two groups: Group 1 for undergraduate students and Group 2 for the graduate or non-matriculated students. After you finish your lab, review other people's games, and select one person (other than yourself), who you think made the best game. The instructor will evaluate your work (10 points) but you will receive extra 2 points, if your game was chosen as the best game. All the students must vote for someone! (Your lab will not be graded, until you finish your lab and survey.) The voting must be done by the next Wednesday (before the Midterm Exam), so make sure to submit your assignment before it, if not earlier.

Description:

The player drag and drop on the screen to roll a ball. The play wins if the ball gets in a cup and loses if it misses.

Implementation Outline:

1. Preparation: You need two images. One is for the background, and the other is for the ball.
2. Interface: Display the images in appropriate positions.
3 Logic: Move the ball based on the drag-and-drop action and judge a hit and miss.

Lab Instructions:

1. Preparation: Background Image and Ball Image

You need two image files, one for the background, and the other is for the ball. The recommended size for each image is as follows (but you don't need to follow the recommendations):

Background: 300 x 480
Ball: 50 x 50

Web browsers typically support JPEG, GIFF, and PNG files. You need to use GIFF or PNG, in order to make part of the ball's image transparent. You can do this by using Photoshop. If you don't know how to do it, ask for help.

2. Interface

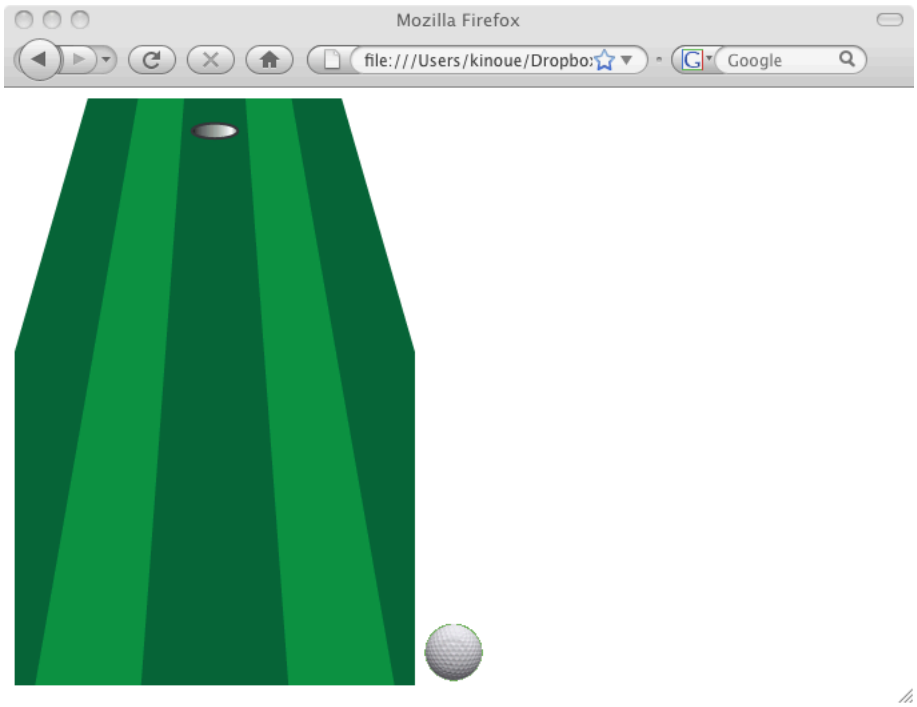
Now, create a new file on your text editor, and save it with a name "Lab6.html" (or whatever you like). As always, insert <script> and <body> elements.

2.1) Displaying images

In the body element, insert two <IMG> elements display the images. At this point, we don't worry about the layout, so just put them consecutively. Use `id` attributes so that JavaScript can identify the elements later. For example, I would say:

```
<img id="bgImage" src="green1.gif">
<img id="ballImage" src="golfball.gif">
```

Save the file and open it on Firefox. It should show the images.



3. Logic

3.1) Modeling the objects

Now, create three global variables in the <script> element to represent the background, the ball, and the cup. You can name them whatever you like, but I will call them `gGreen`, `gBall`, and `gCup`, respectively. Each of them should be defined as a JavaScript object and have the following properties:

- `gGreen:  width, height`
- `gBall:  x, y, radius`
- `gCup:  x, y, radius`

Assign / initialize those properties with appropriate values. **Remember:** the x and y properties of `gBall` and `gCup` represent the coordinate of the center of each object.

3.2) Arrange the image locations

In the <script> element, define a function called `updateDisplay()`. This function will rearrange the locations of images according to the objects' properties. Following are the step-by-step processes of the function:

1. Declare a variable `myBgImage` (or whatever you like), and initialize it by obtaining the background <IMG> element from the document object using the id `bgImage`. In my case:

   ```
   var myBgImage = document.getElementById('bgImage');
   ```

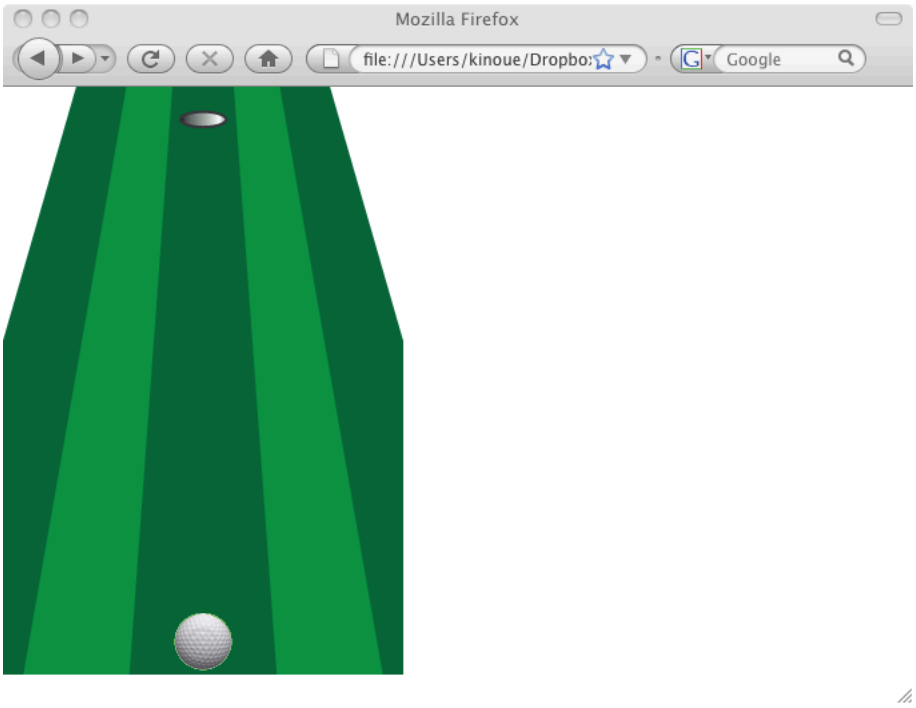
2. Change the following properties of the `myBgImage` object with an appropriate values:

```
myBgImage.style.position
myBgImage.style.top
myBgImage.style.left
myBgImage.style.zIndex
```

3. Repeat the step 1 and 2, but with the `ballImage` element. You need to some calculations based on the `gBall` object in order to determine the right values for `top` and `left` properties.

Once the function is defined, set the `onLoad` attribute of the <body> element to call the function, so that whenever the page is loaded, the function will be called.

Save the file and open it on Firefox. It should now show the images in the right locations.



3.3) Testing Moving the Ball

Define two functions, `hitIt()` and `moveIt()`. The `hitIt()` function should start a timer so that it will call the `moveIt()` function in every 200 milliseconds. By now, you should be familiar with how to start a timer, so I am not going to tell you how to do it. ☺ Just make sure to use a global variable, so that you will be able to stop the timer later.

In the `moveIt()` function, you need to do two things:

1) Obtain the ball object, and change the y (and/or x) coordinate, by subtracting 10 (or whatever you like).

2) Call the `updateDisplay()` function.

Now set the `onClick` attribute of the background image to call the `hitIt()` function.

Save the file and open it on Firefox. Click on the background image. The image of the ball should start moving!

3.4) Displaying the image as a background.

The logical thing to do as the next step would be to work on moving the ball by drag and drop. Unfortunately, we have to do some thing to take care of before moving on.

At this point, the background image is displayed using the <img> element, which itself can be drag and dropped. (Try drag and drop on the background image.) This is not good, because we want to drag and drop the ball (well, sort of), and have the background image stay as it is. The only way I could find is to display the image as a background of something. And in this lab, this something turned out to be an empty table. (There may be better ways to do it, the <table> element is one of the most often used elements in Web pages, so it is good to know how to use it anyway.) The basic structure of <table> elements is the same as any table like spreadsheets (e.g. Excel) or tables in MS Word documents: they have rows, columns and cells. An HTML table is organized by rows (tagged <tr>) and data cells (tagged <td>) and goes like below (You can type them and see how it works, but you don't have to.):

```
<table>                    <!-- This is the start of the table -->
<tr>                       <!-- Row #1 starts here -->
    <td>Data 1-A</td>
    <td>Data 1-B </td>
    <td>Data 1-C</td>
  </tr>
 <tr>                      <!-- Row #2 starts here -->
    <td>Data 2-A</td>
    <td>Data 2-B</td>
    <td>Data 2-C</td>
  </tr>
</table>                   <!-- End of the table -->
```

For our purpose, we use a <table> element with only one row and one empty data cell to replace the current <img> element. Use background attribute instead of the src attribute. It should look like as follows:

```
<table id=”bgImage” height="440" width="300" background=”green1.gif”>
  <tr><td></td></tr>
</table>
```

Notice that I used the height and width attributes, because otherwise, the size of the table will be automatically adjusted to invisibly small (because it has no data inside). This is not a very sophisticated approach, but hey, it does the trick.

Save the file and open it on Firefox. It should look all the same as the previous iteration.

3.5) Testing Drag and Drop

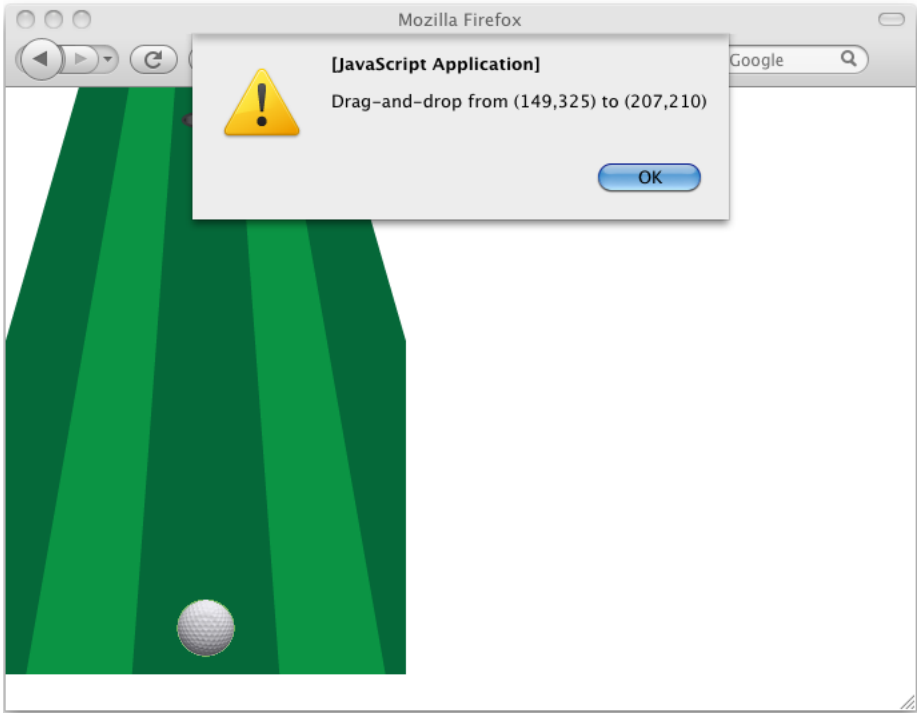
Now we are ready for testing the drag and drop action. Let's add two attributes to the <table> element: `onMouseDown` and `onMouseUp`, each of which should call "`startDragging(event);`" and "`finishDragging(event);`". Don't worry about where those "`event`" parameters come from. They are mysterious like that.

Define four global variables: `gMouseDownX`, `gMouseDownY`, `gMouseUpX`, and `gMouseUpY`, then define the two functions `startDragging(event)` and `finishDragging(event)`, and assign the position of the mouse to those global variables accordingly. For example, the definition of `startDragging(event)` should look like as follows:

```
function startDragging(event)
{
  gMouseDownX = event.clientX;
  gMouseDownY = event.clientY;
}
```

The definition of the `finishDragging(event)` should be similar, but let's insert an alert message to make sure all the assignments are done properly. Say something in the alert function and display all the four global variables.

Save the file and open it on Firefox. Drag and drop on the image, and you should see the alert message you defined.



3.6) Moving the ball using the direction

In order to move the ball in the direction of dragging, you need to change the definition of `moveIt()`, which was defined earlier. Add the following two processes in the function definition:

1. Define a local variable `myDirection`, and initialized it with the direction of dragging, calculated based on the four global variables just defined. Remember, the direction is defined as: (the distance of the mouse moved in the X axis) / (the distance of the mouse moved in the Y axis).

2. Change the `x` property of the `gBall` object, based on the change in the `y` property in the function and the `myDirection` variable.

Call the `hitIt()` function in the `finishDragging(event)` function.

Save the file and open it on Firefox. Drag and drop on the image, and you should see the ball move to the direction of the mouse.

3.7) Final Touch: Hit or Miss Judgment

Now, let's include logic to judge a hit or miss in the definition of `moveIt()`. In the function definition of `moveIt()`, add the following processes:

1. Insert an if statement, to judge if the ball is in the cup. If so 1) clear the timer; 2) hide the ball; and 3) display a message. Remember, the condition of the ball's position to be in the cup is:

   gCup.x – gCup.radius ≤ gBall.x && gBall.x ≤ gCup.x + gCup.raduis &&

   gCup.y – gCup.radius ≤ gBall.y && gBall.y ≤ gCup.y + gCup.radius

   Again, this is not very strict. If you would like, you can make it different (only in a better way, though.)

   To hide the ball, set the `ballImage` element's property `style.visibility` to `"hidden"`.

2. Insert an if statement, to judge if the ball is out of the background image. If so 1) clear the timer; 2) hide the ball; and 3) display a message. The condition of the ball's position to be out of the background is:

   gCup.x < 0 || gCup.y < 0 || gCup.x > gGreen.width

Save the file and open it on Firefox. Drag and drop on the image, and you should see the game is now complete. **Congratulations! Now you can move on the "Submission" Section in the next page**.

(There are no extra tasks for grad students this time. You are very welcome.)

Submission

As always, submit your work to MySite. The Instruction of MySite is found at http://its.syr.edu/mysite/.

You will only need to submit the URL to page you created to the discussion board on the ILMS. Create a hyperlink, if you would like. For example, I would submit:

http://kinoue.mysite.syr.edu/IST400-600/MiniGolf.html

Grading Criteria

The lab assignment must be submitted by the end of Tuesday Mar. 9th. The grading will be based on the following criteria:

- Presentation: 2 points
- JavaScript Syntax: 2 points
- HTML Syntax: 2 points
- Event handling 2 points
- Defining objects 2 points

Late submission will be accepted within one week from the original due date, with 2 points deduction. 2 points will be deducted for improper submissions (e.g. not using MySite) or for not submitting the peer survey as well.