IST400/600 Scripting for Games, Spring 2010
Lab 9
Instructor: Keisuke Inoue

General Description:

In this lab, you will create components of a scrolling game that were discussed in the previous class. The following instructions will help you creating them, 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:

There will be no peer survey this week. Let us wait until we finish the whole game.

Implementation Outline:

A scrolling game is a video game in which the game scene is viewed from the side-view or top-view angle, and the characters move from the left to the right or from the bottom to the top. Classic examples include *Super Mario Brothers*, *Megaman* or (very classic) *Space Invaders*. In this lab you will implement this game with three stages:

1. Scroll foreground images continuously.
2. Displaying Obstacles
3. Detect Collisions

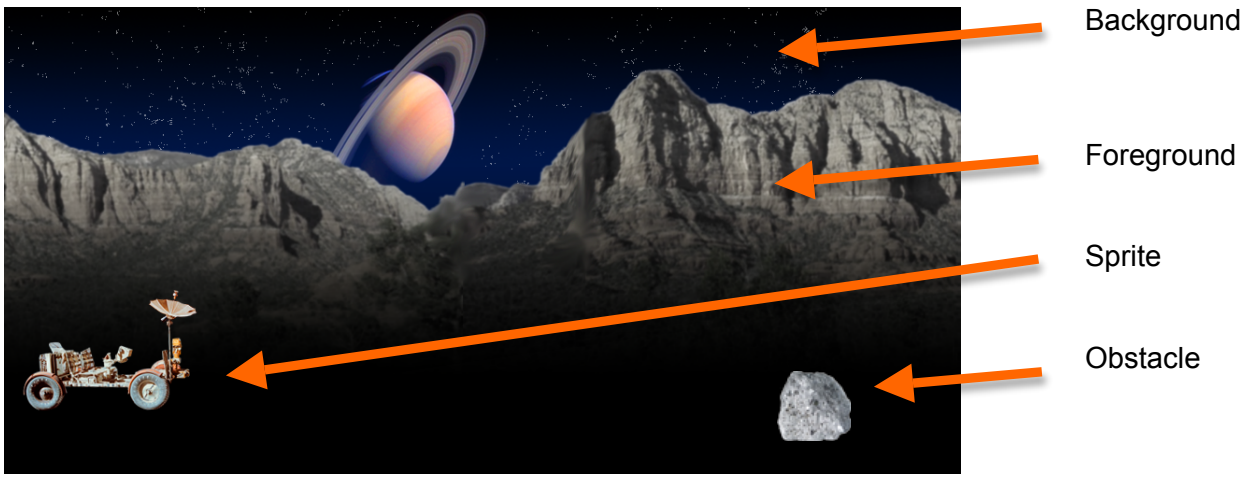
Lab Instructions:

0. Preparations

0.1) Images

This lab assumes you have at least four separate images: background (background.png), foreground (foreground.png), a sprite (sprite.png), and an obstacle (rock.png). The file names are not important, as long as you can keep track of which one plays which role. If you don't have appropriate images, you need to go back to Lab 7 or modify your Lab 7 outcome files.

Below is an example of three images that are used for this lab, over-laid all together.



0.2) Foreground Image

This lab also assumes your foreground image is 1) continuous and 2) has at least twice as wide as the viewable area. 1) does not matter for the grade, but 2) matters, because otherwise, it won't be endless scrolling! You can achieve 2) easily in two ways: i) make an image which is the twice the size of the current image. (By copying-and-pasting the original image, you can easily achieve that. You can use "reflection" or other techniques to make the image look more naturally continuous, but that is not a requirement.); ii) making your viewable area (clipped area) smaller. Obviously ii) is much easier, but make the game look bad, so not recommended, unless you already have a big foreground image.

Below is an example of a continuous image, created from the Nate's original image.

0.3) Lab 7 Files

This lab will be done on top of what you have created in Lab 8. So open your "Lab8.html" with your text editor, and save it as a new it with a name "Lab9.html" (or whatever you like).

1. Scrolling Foreground Image Continuously

1.1) Specifying the new image

Now let's change the source of the foreground image to the new, larger one that you created. (If your image was already large enough, you can skip this step, obviously.)

1.2) Modifying the updateGame() function

Remember that this function moves the foreground image to the left (or right or top or bottom) by changing the `fg.style.left` (or `fg.style.top`) property. You need to do only two things here. (Unless you did the grad. student assignment last time, in which case you need to do only one thing!)

The first step is, after the statement you changed the value of the `fg.style.left` property, insert another statement to change the same property, based on a condition. In other words, insert an statement that moves back the foreground image to the starting position, if the foreground image went too far (and ran out of the image in the viewable area.)

The next step is to modify the statement that changes the `fg.style.clip` property (if you haven't done so.) This should work the same as the previous version, but simplifying the logic using the `fg.style.left`.

```
fg.style.clip = genRect(    0,
                            gs.viewWidth - parseInt(fg.style.left),
                            gs.viewHeight,
                            - parseInt(fg.style.left));
```

Do understand why it works? What happens if you don't do this modification?

At this point, save the file and reload it on Firefox. The scrolling should be endless now.

2. Displaying Obstacles

2.1) Creating a basic structure

2.1.1) Adding properties to the global gGameState variable
First, let's add some properties to the global `gGameState` varible to keep track of everything. Below are the properties. Make sure to initialize them appropriately!

`gGameState.obsList`:	Keeping track of ID's of obstacles. Initialize it as an empty array.
`gGameState.obsNum`:	Counting number of obstacles generated so far. Initialize it with 0.
`gGameState.obsSpeed`:	Specifying the number of pixels per scroll that obstacles move.
`gGameState.obsInterval`:	Specifying the interval to create obstacles.

You may also want to create the following properties, depending on the direction of scrolling (for example, if your game scrolls from the top to the bottom, you will want `obsStartingTop` rather than `obsStartingLeft`. Again, make sure to initialize them appropriately.

`gGameState.obsStartingLeft`:	The left position of the obstacle, when it first appears.
`gGameState.obsTopLimit`:	The top position that obstacle can appear at the top limit.
`gGameState.obsBottomLimit`:	The top position that obstacle can appear at the bottom limit.

2.1.2) Defining the createObstacle() function
This is the function you create obstacles. For now, let's just define it as a testing function, and call alert() (with whatever the message) in the function definition.

2.1.2) Calling the createObstacle() function
In the updateGame function, call the `createObstacle()` function every `gGameState.obsInterval` times the function is called. You will need the `gGameState.scrollNum` property, which was defined in the previous lab. (If you are a grad student, you might have deleted it, in which case you need to bring it back.) Remember how to do it?

At this point, save the file and reload it on Firefox. Start scrolling, and you should see the alert message every once in while.

2.2) Defining the createObstacle() function (for real)

2.2.1) Declaring book-keeping objects for obstacles.
As we did in the MiniGolf game, we need to keep track of some information about obstacles we are about to display. For now, we only need to keep track of the elements' IDs, but soon, there will be more. So, we want to create an object for each obstacle, and save it in the `gGameState.obsList` array. In the `createObstacle()` function definition, insert the following steps:

1. Declare a new local variable `newObs`, and initialize it as an object, with one property: `id`, which should be initialized with `gs.obsNum`.
2. Add the `newObs` object to the `gs.obsList` array, using push() method.
3. Increase the `gs.obsNum` variable by one.

2.2.2) Creating an HTML element
Now, you will create an HTML element counterpart of the object that you created. In the `createObstacle()` function definition, below the segment you created in 2.2.1), insert the following steps:

1. Declare a new local variable `obs`, and initialize it by calling `document.createElement('img')`.
2. Initialize the property `obs.style.position` with `'absolute'`.
3. Initialize the property `obs.style.zIndex` with 2 (or higher value.)
4. Initialize the property `obs.style.left` with `gGameState.obsStartingLeft`.
5. Initialize the property `obs.style.top` with 400 or so just for testing for now.
6. Initialize the property `obs.style.src` with the path to the image file.
7. Set the `id` attribute of `obs` to `'rock' + obs.id`, using the `setAttribute()` method.
8. Add the object to the document body by calling `document.appendChild(obs)`.

(The steps 4 and 5 are different, if you are working on a vertical scroller, but you should get the idea, by this time.)

Save the file and reload it on Firefox. Start scrolling, and you should see one obstacle image appears in the position specified. (Actually there are more obstacles created, as you keep scrolling, but since the position of the image stays the same, you can see only one image.)

2.2.3) Positioning obstacles in random positions.
Now, change the assignment to the `obs.style.top` property, so that it will be a random value between `gs.obsBottomLimit` and `gs.obsTopLimit`. Remember: `Math.random()` returns a random value between 0 and 1.

Save the file and reload it on Firefox. Start scrolling, and you should see obstacles appear in a random position once in while.

2.3) Scrolling the obstacles

This function scrolls all the obstacles one by one. Define the function by inserting the following steps:

1. Create a for loop, that goes through the `gs.ObsList` array.
2. Inside the for loop, declare a local variable `obs`, and initialize it by obtaining the counterpart HTML element object using `document.ElementById()`. What is the argument for the method?
3. (Still inside the for loop) check if `obs` is not null, and if not, do the following:
    a. Change the position of `obs` using `gs.obsSpeed`.
    b. If `obs` has gone too far (e.g. beyond the left end of the browser), remove the element from the document body by calling `document.body.removeChild()`.

By the way, do you know why we need to check if the object is null?

Once again, save the file and reload it on Firefox. Start scrolling, and you should see obstacles appear in a random position once in while. The obstacles should be now scrolling, too.

3. Detecting Collisions

We will work on more about handling collisions next week. In this lab, we will work on just detecting it.

3.1) Defining the checkCollision() function.

Now define a function, called `checkCollision`(), that takes two parameters, say `e1` and `e2`. This function sure return *true* if the following conditions are met:

1. The left-most coordinate of e1 (say e1x) is smaller than the right-most coordinate of e2 (say e2x + e2w).

2. The left-most coordinate of e2 (say e2x) is smaller than the right-most coordinate of e1 (say e1x + e1w).
3. The top coordinate of e1 (say e1y) is smaller than the bottom coordinate of e2 (say e2y + e2h).
4. The top coordinate of e2 (say e2y) is smaller than the bottom coordinate of e1 (say e1y + e1h).

Return *false*, otherwise.

3.2) Calling the checkCollision() function

Insert the following steps in the `updateObstacles()` function.

1. In the beginning of the function, declare a local variable `sp` (or something) and initialize it by obtaining the main sprite object from the document.
2. Inside the for loop, call the `checkCollison()` function with the arguments `sp` and `obs`. in the end. If the returned value of the function call is true, call the `alert()` function to notify the player.

Bonus Steps (4 points):
Modify the `updateObstacles()` function further, so that when the collision is detected, not only the player is notified by also the game stops the scrolling (2 points), and goes back to the state before the game started (2 points).

Save the file and reload it on Firefox. Start scrolling and try if collisions are detected. Congratulations! You finished this lab!

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/Lab9.html

Grading Criteria

The lab assignment must be submitted by the end of Tuesday April 13th. The grading will be based on the implementation of the following features:

- Scrolling the foreground image endlessly: 2 points
- Displaying obstacles with a set interval: 2 points
- Positioning obstacles in random positions: 2 points
- Scrolling obstacles: 2 points
- Detecting collisions with obstacles: 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.