IST400/600 Scripting for Games, Spring 2010
Lab 8
Instructor: Keisuke Inoue

General Description:

In this lab, you will create the 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. <u>Do not copy-and-paste any resources other than you created.</u>**

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:

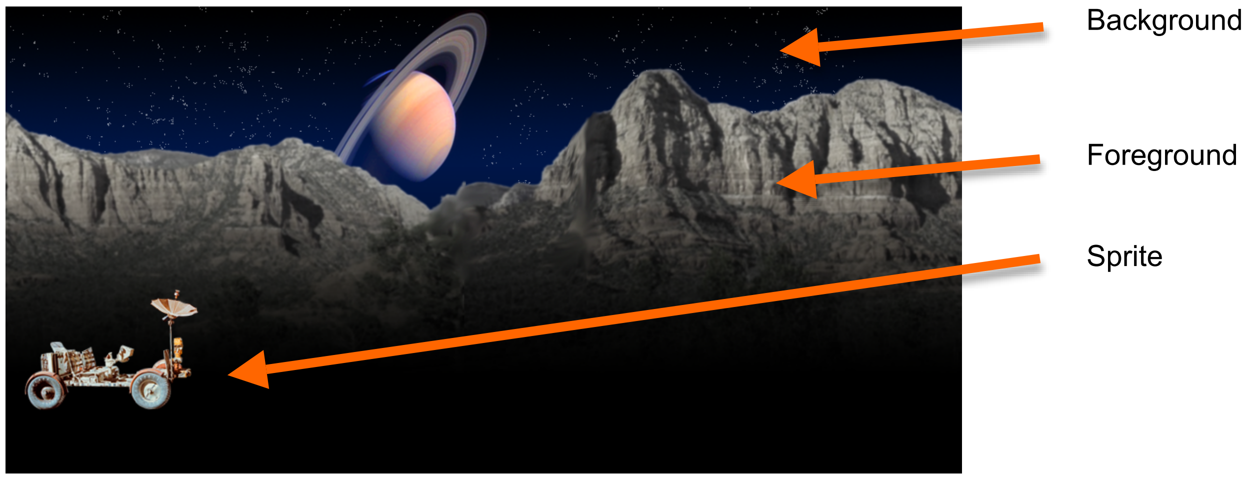
1. Lay out the image elements.
2. Scroll the foreground image(s).
3. Control the main sprite from the keyboard.

Lab Instructions:

0. Preparations

This lab assumes you have three images: background (background.png), foreground (foreground.png), and a sprite (sprite.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.



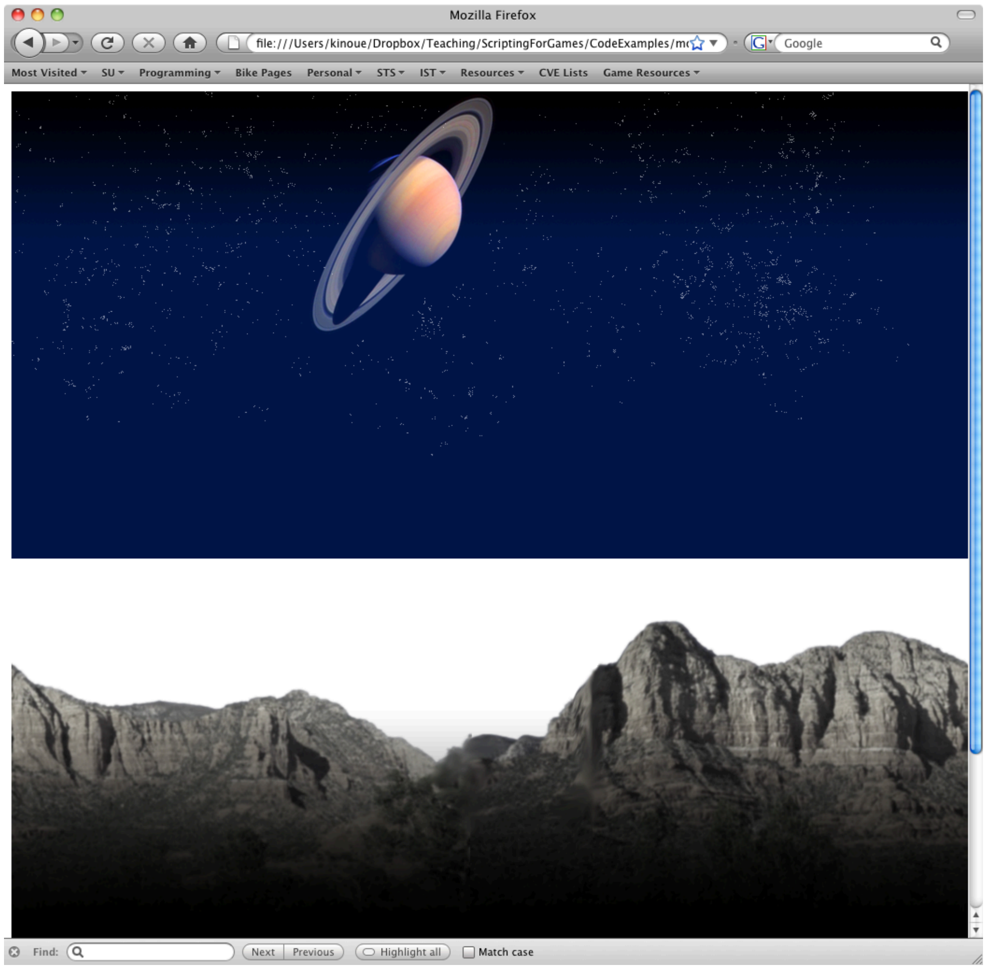
1. Laying out the images

Create a new file on your text editor, and save it with a name "Lab8.html" (or whatever you like). As always, insert <script> and <body> elements.

1.1) Displaying images

In the <body> element, insert three <img> elements to display the images. At this point, we don't worry about the layout, so just put them consecutively. Set `id` attributes to `"bg"`, `"fg"`, and `"sprite"` (or what ever you like) respectively, so that JavaScript can identify each element.

Save the file and open it on Firefox. It should simply show the images as below: (Oops, the page was too long to fit in the screen.)



1.2) Arranging the image locations

In the <script> element, define a function called `initalizeGame()`, which arranges the locations of images. For each of the three images, do the following:

1. Declare a variable, for example, myBackground (or something like that), and initialize it by obtaining the corresponding <img> element using the `document.getElementById()` method. For example:

   ```
   var myBackground = document.getElementById('bg');
   ```

2. Then, change the following properties of the `myBgImage` object with an appropriate values:

   ```
   myBackground.style.position
   myBackground.style.top
   myBackground.style.left
   myBackground.style.zIndex
   ```

Now, set the `onLoad` attribute of the <body> element to call `initalizeGame()`.

Save the file and reloade it on Firefox. It should show the all the images laid-out properly (as in the first image example of this document.)

2. **Scrolling the foreground**

2.1) Setting up

Before we start implementing all the actions, we will first prepare some information for the scrolling to happen.

First, in the <script> element, create a global variable called `gGameState` (you can name it with a shorter name, if you prefer.) to contain all the setups and/or status of the game. Initialize it as a new object. (Do you remember how?) Then, in the beginning of the `initalizeGame()` function, initialize the following properties with appropriate values:

o `gGameState.scrollCount`:	How many times the screen has been scrolled.
o `gGameState.scrollSpeed`:	The speed of scrolling in the number of pixels per scroll)
o `gGameState.scrollInterval`:	The scrolling interval in milliseconds
o `gGameState.viewWidth`:	The width of the viewable area in pixels
o `gGameState.viewHeight`:	The height of the viewable are in pixels
o `gGameState.spriteSpeed`:	The speed of the sprite in the number of pixels per key press

Phew, that's a lot, but you should all remember from the class (at least vaguely), how these properties come handy later in the implementation.

2.2) Using the Timer

Define a function `updateGame()`function that does the following:

1. Declare a variable `myForeground` (or something like that), and initialize it by obtaining the corresponding <img> element from the document object using `document.getElementById()` method.

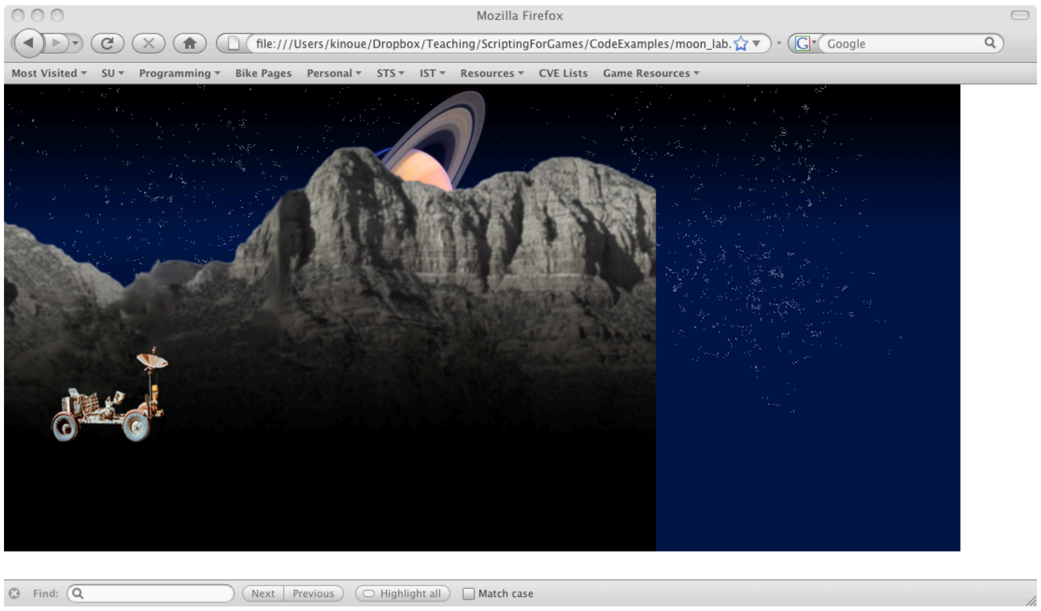
2. Decrease or increase the value of `style.top` or `style.left` property by `gGameState.scrollSpeed.` Don't forget to use the `parseInt()` function!

Now define a global variable `gTimer`, and a function `startGame()` that starts a timer so that the `updateGame()` function will be called every `gGameState.scrollInterval` milliseconds. Set the `onClick` attribute of the sprite, so that if you click the sprite, the game will start.

Extra Assignment for Graduate Student:

Modify the startGame() function so that eveytime the player click the sprite, it will either start (or restart) or pause the game.

Save the file and reload it on Firefox. Click on the sprite. It should scroll the foreground, but the foreground will disappear at it scrolls!



2.3) Clipping the Image

As we discussed in the class, we will fix the problem by displaying only a small portion of the image. In order to so, modify the functions we defined so far as following:

1. First, define a helper function, called `genRect()`. This is just for our convenience, so if you don't want to do it (and you know what you are doing), you can skip this. This function just returns a string to define a rectangular area in CSS, that can be set to the `style.clip` property of an HTML element, based on four parameters. Below is the whole code:

   ```
   function genRect(top, right, bottom, left)
   {
      return "rect(" + top + "px," + right + "px, " +
                            bottom + "px, " + left + "px)";
   }
   ```

2. Now in the `initiliazeGame()` function, initialize the `style.clip` property of the background and foreground elements, using the `gGameState.viewHeight` and `gGameState.viewWidth`. For example (I assigned `gGameState` to a local variable `gs`, to save typing.):

   ```
      myBackground.style.clip = genRect(0, gs.viewWidth, gs.viewHeight, 0);
   ```

   Also in the `initiliazeGame()` function, initialize the `gGameState.scrollCount` property to 0.

3. In the `updateGame()` function, modify the `style.clip` property of the foreground element as follows:

   ```
   myForground.style.clip =
      genRect(   0,
                 (gs.scrollCount * gs.foregroundSpeed + gs.viewWidth),
                 gs.viewHeight,
                 gs.scrollCount * gs.foregroundSpeed);
   ```

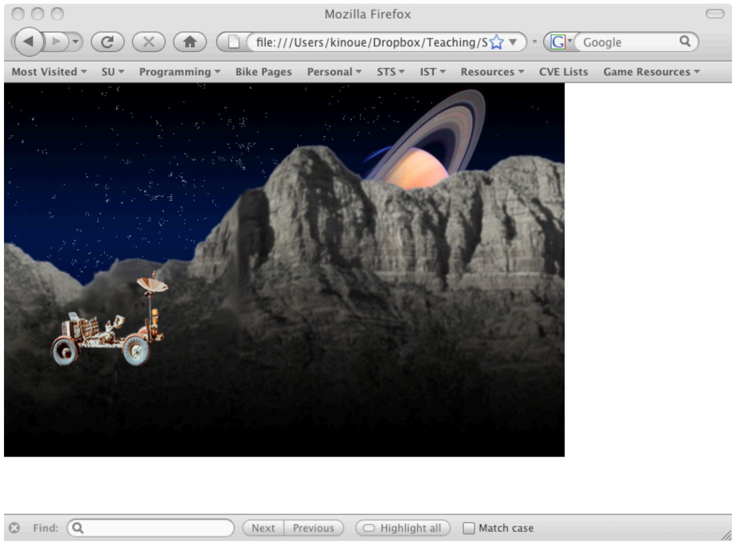
   Increase `gGameState.scrollCount` by 1.

   Do you understand why it works?

Bonus Assignment (additional 2 points)

You can specify the clipping area, without using `gGameState.scrollCount`, if you understand why the clipping works. Come up with this simpler method and incorporate it in the step above. (HINT: Use the `myForeground.syle.left` property.)

Save the file and reload it on Firefox. Make sure the images are clipped properly. Click on the sprite and the foregrounds should scroll without disappearing (at least for a while).



3. Controlling the Sprite from the Keyboard

We are finally at the last stage of today's lab! (Is it only me who is feeling this lab is taking longer than usual…?)

3.1) Setting up onKeyPress event

In order to take the events from the keyboard, set the `onKeyPress` attribute to "`moveSprite(event)`" of the <body> element. In this way, every time the player press any keys, the `moveSprite()` function is called. (The name `moveSprite()` is not a good one, because you may want to do something else other than simply moving the sprite (e.g. shooting a missile, quitting the game, etc.) but for now, this will do for us.

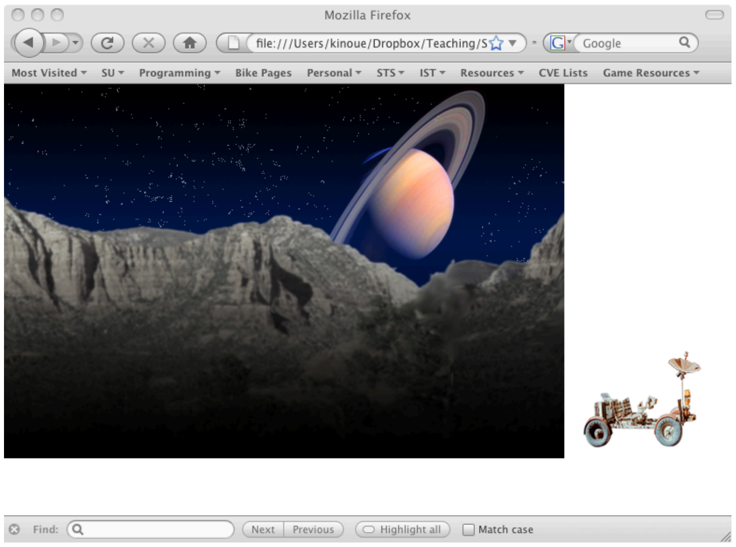
3.2) Defining the event handler.

Define the `moveSprite()` function, based on your need. Your sprite doesn't need to move to the all directions. You can use the switch statement if you would like, or you can stick with the old-style if-else statement. In either case, test the `event.keyCode` property and change the `style.left` or `style.top` property of the sprite accordingly. Here are the key codes for the arrow keys:

37:	left arrow key
38:	up arrow key
39:	right arrow key
40:	down arrow key

Increase or decrease the values of the properties using `gGameState.spriteSpeed.`

Save the file and reload it on Firefox. Press the arrow keys, and make sure the sprite is moving correctly.



3.3) Checking if the sprite is allowed to move.

Sometimes, a sprite is not allowed to move to a certain position. This can be because of there are walls or obstacles in the game scene or, in our case, because we simply don't want the sprite to go outside the game scene (as in the screenshot above).

This can be prevented by incorporating a function, let's called it `canMove()`, which checks if a given position is legal for a sprite.

Defined the function `canMove()` with two parameters `myLeft` and `myTop` (or whatever you like). In the function definition, do the following:

1. Define a local variable `mySprite`, and initialize it by obtaining the sprite element from the document. (This can be parameterized, and specified by the `moveSprite` function, if you would like.)

2. Return *true*, if all the following conditions are met:
   - `myLeft` is larger than 0,
   - `myTop` is larger than 0
   - `myLeft + mySprite.width` is smaller than `gGameState.viewWidth`.
   - `myTop + mySprite.height` is smaller than `gGameState.viewHeight`.

3. Return *false* otherwise.

Modify the `moveSprite()` function so that it tests with the `canMove()` function before it actually changes the position of the sprite. In order to make the code clean, it may use some reorganization of the codes (just like I did in the class). For the purpose of this lab, this reorganization is not necessary.

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/Lab8.html

Grading Criteria

The lab assignment must be submitted by the end of Tuesday April 6th. The grading will be based on the implementation of the following features:

- Image layout:	2 points
- Clipping:	2 points
- Scrolling:	2 points
- Keyboard Event	2 points
- `canMove()`	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.