IST400/600 Scripting for Games, Spring 2010
Lab 10
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 for this lab. Let us wait until we finish the whole game.

Implementation Outline:

1. **Handling collisions ver. 2: Re-initializing the game**
2. **Handling collisions ver. 3: Handling multiple lives of the character**
3. **Displaying scores and lives**

Lab Instructions:

0. Preparations

This lab will be done on top of what you have created in Lab 9. You should have submitted your Lab9 before you submit this one. So open your "Lab9.html" with your text editor, and save it as a new it with a name "Lab10.html" (or whatever you like).

1. Re-initializing the game

In this step, you will improve the collision detection mechanism that you defined last week. (This was a bonus step in the previous lab, so if you have done it already, you can skip this step.) What you need to do is to identify the code segment that detects a collision (which must be an if-statement), in the `updateGame()` function, and insert the following three steps to the if clause: 1) strop the timer; 2) show the message "Game Over". 2) clear all the obstacles on the screen; and 3) initialize all the variables.

1.1) Stop the timer

If your `startGame()` function is defined to start AND stop the timer (depending on the situation), all you need to do is to call the `startGame()` function. If not, you need to modify the function so that the function starts the timer when the timer is not started, and stops the timer otherwise. In other words, make sure the code of your `startGame()` is doing the following:

1. Test if `gTimer` is null.
2. If so, start a timer, so that it will call the `updateGame()` function with an interval `gGameState.gameInterval`.
3. If not, stop the timer, and assign null to `gTimer`.

1.2) Show an alert message

(You have done this already.)

1.3) Clear all the obstacles

For this, you need to define a new function, let's call it `clearObstacles()`, before you use it. Clear obstacles have to do basically two things. 1) Go through the `gGameState.obsList` array, and for each obstacle object, remove the corresponding HTML element from the document body. The pseudo code for this step is:

1. Create a for loop, that goes through the `gGameState.obsList` array.
2. For each element in the array, define a object, say, `obs`, which represents the corresponding HTML element. (You need to use the `document.body.getElementById()` method to obtain the object.) If the corresponding element is found (i.e. is not null), remove it from the document body.
3. After the for loop, initiaze the `gGameState.obsList` array, so that it becomes empty.

HINT: If you can't figure out how to implement it, take a look at the for loop in the `updateGame()` function. All the steps need to define the `clearObstacles()` function are there.

1.4) Initialize the variables.

Call the `initializeGame()` function (and that's it!).

At this point, save the file and reload it on Firefox. Start the game and make the main character to collide with an obstacle. The game should be initialized and look just like before starting the game.

2. Multiple lives

In this step, we modify the game so that the character has multiple lives in one game. (In other words, the player can continue to play the game even after the character "dies" for a few times. Basically, there two things you need to do: 1) define a variable (property) to keep track of the number lives left; and 2) handle a collision based on the number of lives left.

2.1) Keeping track of the number of lives

In the `initializeGame()` function, define a new property of the `gGameState` global variable, say, `gGameState.lifeNum`. Initialize it with whatever the number you want.

2.2) Modifying the initializeGame() function

(This section is slightly different from the lecture. So be careful.)

Before moving the second step, we need to modify the `initializeGame()` function. This function has been pretty useful, because there are many occasions that we want to initialize the state of the game, and this function does everything for us. However, this function lacks flexibility, in that it really initializes <u>everything</u>. Now, we want a function that initializes pretty much everything, but <u>not</u> the game score and number of lives left. So we are going to isolate those variables. Here is how we do it:

1. Delete the statements where they initialize the `gGameState.scrollNum` and `gGameState.lifeNum` properties.
2. Now, rename the `initializeGame()` function to something different, say, `clearGame()` (or you can come up).
3. Then define a new `initializeGame()` function, to do the following:
    1. Initialize `gGameState.scrollNum`.
    2. Initialize `gGameState.lifeNum`.
    3. Call the clearGame() function.

The idea is that now we have two very similar functions: 1) `initializeGame()`, which initializes everything; and 2) `clearGame()`, which initializes everything but `gGameState.scrollNum` and `gGameState.lifeNum`.

2.3) Handling Collisions based on the number of lives left.

Now, identify the code segment that handle the collision detection in the `updateGame()` function (which must be an if-statement), then insert the following logic in the if clause:

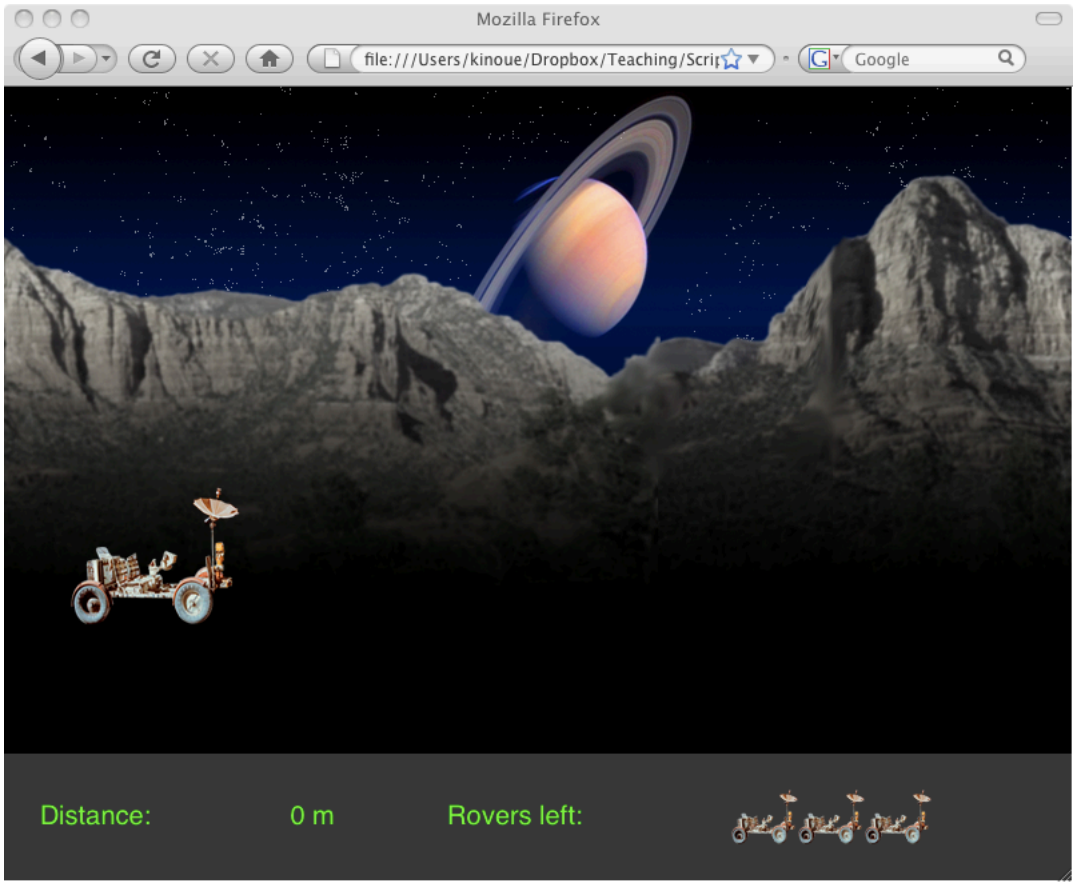
1. Test if `gGameState.lifeNum` is less than or equal to 0.
2. If so, proceed with the previous logic (re-initializing the game).
3. If not, do the following:
    1. Stop the timer.
    2. Display a different alert message, something not saying “Game Over”.
    3. Clear the obstacles.
    4. Decrease the `gGameState.lifeNum` variable by one.
    5. Call the `clearGame()` function. (Here, you don't want to call `initializeGame()` function, because it will initialize the score and the number of lives left, too.)

Save the file and reload it on Firefox. Start the game and make sure everything is working so far.

3. Displaying scores and lives

In this step, we will work on displaying the score and number of lives left. We will use an HTML table to display them and update them whenever needed.

Although the example in the class showed scores and lives left displayed on top of the viewable area, you can display it anywhere you want in the browser’s window. You just need to adjust the coordinate accordingly. For example, the following is an example of different positioning.



3.1) Defining an <table> element

In the <body> element, insert a table. Set the `id` attribute of the table to "`score_board`" (or something like that.)

Then insert a table row element and at least two table cell elements: 1) with the ID attribute set to "`score_sb`" to display the score (obviously) and 2) with the id attribute set to "`life_sb`" to display the number of lives left. Add table cells to display labels such as "Score:" or "Distance:" or "Number of rovers left:", etc. as you like.

3.2) Positioning the table

Position the table in the `clearGame()` function, by following the steps below:

1. Declare variable, named `sb`, and initialize it by obtaining the `score_board` element from the document body.
2. Assign the following properties according to your preference:

   - `sb.style.position`
   - `sb.style.zIndex`
   - `sb.style.top`
   - `sb.style.left`

   Optionally, you may want to set the following properties:

   - `sb.style.fontFamily`
     To change the font family. Options include: `"serif"`, `"sans-serif"`, `"cursive"`, `"fantasy"`, `"monospace"`, etc.
   - `sb.style.fontWeight`

Options include: "normal", "bold", "bolder", etc.

- `sb.style.fontSize`
  To set the size of the font (obviously).
- `sb.style.color`
  To set the foreground color. Options are listed in:
  http://www.w3schools.com/css/css_colorsfull.asp
- `sb.style.background`
  To set the background color.
- `sb.width`
  To adjust the width.

3.3) Updating the scores and number of lives left.

Updating the scores and numbers is pretty simple. You can define a function, `updateScoreBoard()` to do this and then call the function at the end of the `updateGame()` function. Below are the steps that should be included in the definition of `updateScoreBoard()`:

1. Obtain the HTML elements, "`score_sb`" and "`life_sb`" from the document.body. Call them `scoreSb` and `lifeSb` respectively.
2. Assign `gGameState.lifeNum` to `lifeSb.innerHTML`.
3. Assign `gGameState.scrollCount` (or `scrollCount`) to `scoreSb.innerHTML`.

The "`score_sb`" element is not important now, since we haven't defined any event to gain scores (e.g. shooting down enemies). So at this point, you can just assign 0, if you would like. But it is more entertaining to show the changing values.

Bonus Steps (4 points):
Modify the code further to indicate the number of lives left by displaying smaller images of the main sprite. (You can to create a smaller image of the main sprite using PhotoShop, or you can use the same image of the sprite, and set the height or width property to shrink it.) Remember that in order eliminate the flickering, you need to modify the code so that it will update the "`life_sb`" element only when the collision is 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/Lab10.html

Grading Criteria

The lab assignment must be submitted by the end of Tuesday April 20th. The grading will be based on the following:

- Definition of initializeGame() function: 2 points
- Definition of clearGame() function: 2 points
- Handling multiple lives: 2 points
- Positioning the table: 2 points
- Displaying & updating the number of lives left: 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.