CS 1100: Web Development: Client-Side Coding / Fall 2016
Lab 11: Exploring jQuery

Goal

- Explore jQuery. Lots of it!

Work together / Submit work individually

This lab is a work-in-pairs exercise: work together, but use 2 laptops. BOTH STUDENTS SHOULD SUBMIT THEIR WORK. Your code can be identical (although it doesn't have to be).

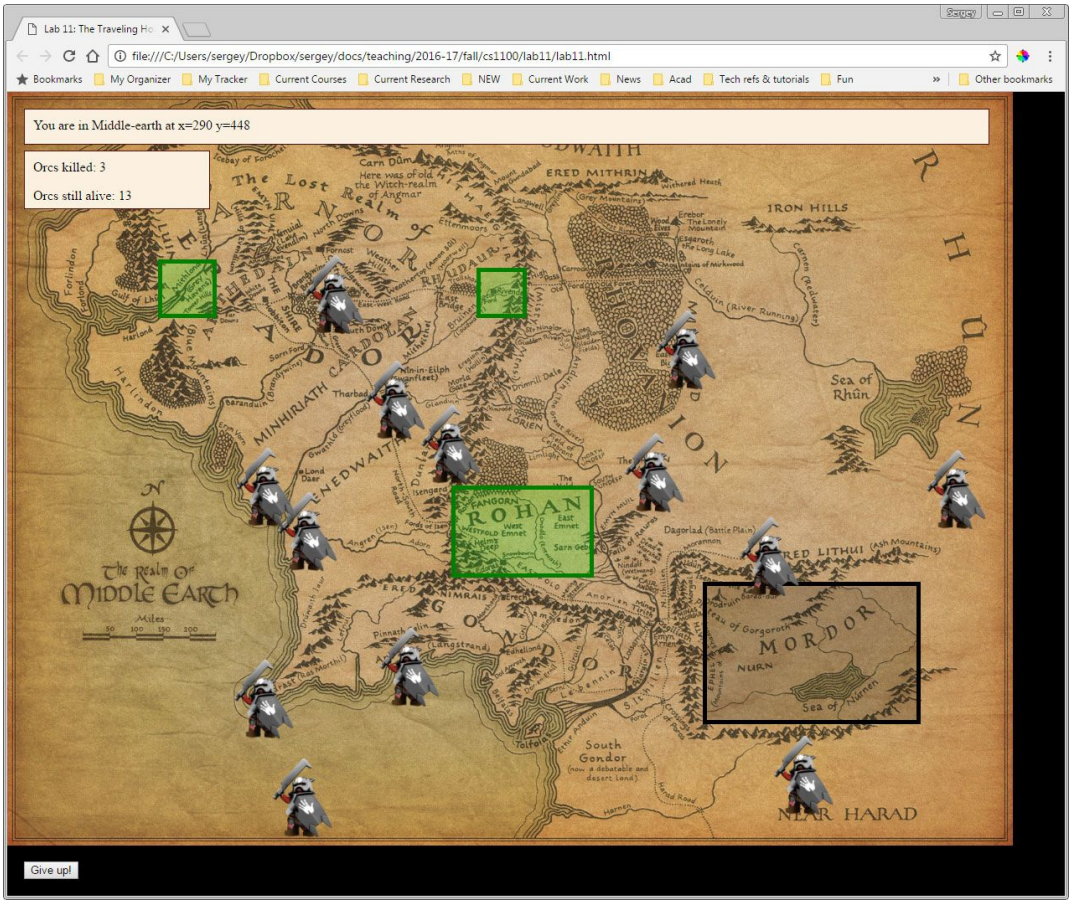
Setup

Download the lab11.zip file and unzip it. You will see a lab11 folder that contains 3 images, a css file and html file, and a js folder that contains the jquery library file.

Your lab11.html file references a lab11.js file - which you need to create. This is where you will put your code.

Description

We will implement a simple game today. First, we will recreate the functionality of The Travelling Hobbit (exercise from lab 8 and homework 7). Then we'll add some basic interactivity, focused mostly on killing orcs. This is what it looks like:



Step 0

Before you start, take a look at a few changes in the html and css (compared to the code you've used before):

- HTML:
  - line 10: a new div containing a logo.png image
  - line 12: the rest of the divs are wrapped in a div with id="game"
  - line 19: a "quit" button is added
  - lines 14-17: the locations divs are given a common class (that's an example of when we need class attributes in addition to id attributes)
  - line 19: a new "score" div added
  - lines 23-24: we have a reference to the jQuery file (we link to a CDN and then, on the next line, check if the jQuery object exists; if not - we add a reference to the local file)
- CSS:
  - the body background is set to black (we'll need that at the very end)
  - some of the selectors are grouped to remove duplicate code
  - line 50: we specify color using the rgba() notation: that way we can specify opacity as the fourth argument: rgba(0, 255, 0, 0.2) is green with opacity=0.2

Step 1

Right now your web page is blank, with a black background. We'll change this in a moment. Make sure you open your browser console sidebar to catch any errors.

1.1. First, add the jQuery "wrapper" to your lab11.js file:

```
$(function() {
    //all of your code goes here
});
```

1.2. Now show your map with all of its contents:

```
$("#game").show();
```

This code creates a jQuery object that wraps an HTML element that we've obtained using the CSS selector "#game". Then it calls the show() method, which changes the CSS display property of the underlying HTML, making it visible. (More details: http://api.jquery.com/show/)

1.3. Now display the message box using a basic animation effect:

```
$("#message").slideDown();
```

(More details: http://api.jquery.com/slideDown/)

1.4. Finally, let's use another animation effect to display all the location divs:

```
$(".location").fadeIn(2000);
```

Wait! Nothing happened! Why? Because the divs were already displayed - so let's hide them first, and only then fade-in:

```
$(".location").hide()
$(".location").fadeIn(2000);
```

The argument for fadeIn is the number of milliseconds (i.e., 2000 = 2 seconds).
But we can write code that is more elegant: instead of the previous code, let's chain our 2 methods: hide() and fadeIn(2000). This code is also more efficient: ***we search the DOM tree for ".location" only once!***

```
$(".location").hide().fadeIn(2000);
```

Notice how the changes were applied to all elements in the matched set: this is called **implicit iteration**: we didn't have to write a loop to access all elements in the set - jQuery does it for us.

(More details: http://api.jquery.com/fadeIn/ and http://api.jquery.com/hide/)

As a result of these steps, your code should look like this:

```
$(function() {

	$("#game").show();
	$("#message").slideDown();
	$(".location").hide().fadeIn(2000);

});
```

Check it in the browser - make sure you see the map, the locations on the map and the message box.

Step 2

Now let's replicate our functionality from lab 8: we want the message box to display a message whenever we are inside one of the locations; at all other times, we want to see our current coordinates.

2.1. We start by adding a simple event handler:

```
$("#rohan").on("mouseover", showRohan);
```

We get a jQuery object using "#rohan" as the selector. Then we call the on() method, passing an event name and a function name as arguments. Then we write our function definition:

```
function showRohan() {
	$("#message").text("You are in Rohan");
}
```

The function gets a jQuery object using the "#message" selector, and then calls the method text(), passing the new text as an argument: that's how we update the text content of a jQuery element.

Try out this code!
(More details: http://api.jquery.com/text/ and http://api.jquery.com/on/)

2.2.Consider the function we've created. the only time we'll need it is when the mouse is over the "rohan" div. We'll never reuse it (there's no other way we can be in Rohan other than having the mouse over it!). Since we don't need to call this function anywhere else, we can make our code much more compact by using an anonymous function. Replace the code you've just written (both the statement and the function definition) with this:

```
$("#rohan").on("mouseover", function() {
      $("#message").text("You are in Rohan");
});
```

Instead of passing a function name as an argument, we pass the actual function definition, skipping the name. Try out your code.

2.3. Let's do some minor refactoring. Consider this: each time we write this: $("our selector"), our program has to search the DOM tree to locate the elements that match our selector. If it's a large html file, it will be an expensive operation. Therefore, we want to do it only once whenever possible.

In our program, we look for "#message" twice, and we are about to add several new instances (for each location). So let's do it once and assign the result to a variable:

```
var $messageBox = $("#message");
```

Now we can change the rest of our program to use our variable instead of searching for the element over and over again:

instead of this: `$("#message").slideDown();` do this:

```
$messageBox.slideDown();
```

Same goes for the body of the anonymous function we've created:

```
$("#rohan").on("mouseover", function() {
      $messageBox.text("You are in Rohan");
});
```

2.4. Now add the same code for the other 3 locations (grayhavens, .rivendell, mordor)

2.5. Finally, let's take care of updating our coordinates. In this case, we pass the event object to our anonymous function. We access the mouse coordinates exactly the same way we did before: e.clientX and e.clientY:

```
$("#map").on("mousemove", function(e) {
      var text = "You are in Middle-earth at x=" + e.clientX + " y=" + e.clientY;
      $messageBox.text(text);
});
```

We're done here! Check your code by moving the mouse.

Step 3

3.1. Let's add an orc to our map! First, add it to your webpage to see how it should look. Place this code inside your "game" div, after the "quit" button:

```
<img src='orc.png' style='width: 100px; position: absolute; left:100px; top: 100px;'>
```

Look at the page. This is what we need to implement in our js file. Now delete the html you've added (we only needed it to see what our goal is).

3.2. Start with this: when the user clicks the map, an orc appears. To do that, let's handle the "click" event:

```
$("#map").on("click", function() {
      var orc ="<img src='orc.png' style='width: 100px; ";
      orc += "position: absolute; left:100px; top: 100px;'>";
      $("#game").append(orc);
});
```

Our anonymous function is called when the "map" div is clicked:

a. It builds an html string containing the <img> tag;
b. creates a jQuery object for the "game" div; and
c. appends the new html to the end of the "gave" div's contents.

Run your code!

3.3. Let's improve this by placing the orc exactly where we clicked the map. For that, pass the event object e to the anonymous function and use the mouse coordinates, just like we did in step 2.5.

```
$("#map").on("click", function(e) {
      var css = "width: 100px; position: absolute; ";
      css += "left: " + e.clientX + "px; top: " + e.clientY + "px";
      var orc = "<img src='orc.png' style='" + css + "'>";
      $("#game").append(orc);
});
```

Run your code! In your console, click the Elements tab and drill down to see the contents of the "game" div. Now start clicking the map to add orcs - you'll see new <img> elements added to your html!

Step 4

4.1. Let's do the same refactoring we did in step 2.3, but this time for game" and "map".

Assign selections to variables:

```
var $game = $("#game");
var $map = $("#map");
```

Now use the variables:

```
$game.show();

$map.on("mousemove", function(e) { ...
$map.on("click", function(e) {...
```

4.2. We can create any number of orcs. However, we should be able to kill them too - because, in the words of Sam Gamgee, "there's some good in this world, Mr Frodo... and it's worth fighting for!" So, let's kill some orc!

We kill orcs by removing them from our html in response to a click. So, we need to add an event handler to each orc we create. For that, our html string (the one that contains the <img> element) should be turned into a jQuery object. Then we can add our event handler.

Alter your $map.on(.... method: (the dark red is the updated code);

```
$map.on("click", function(e) {
      var css = "width: 100px; position: absolute; ";
      css += "left: " + e.clientX + "px; top: " + e.clientY + "px";
      var $orc = $("<img src='orc.png' style='" + css + "'>");
      $game.append($orc);

      $orc.on("click", function() {
            $orc.remove();
      });
});
```

(More details: http://api.jquery.com/remove/)

Run your code - try adding and then removing orcs. Consider this - we have a function in a function in a function!

Step 5

5.1. Let's keep track of how many we've killed and how many we still have to kill.

Create 2 variables and set them to 0 - these are our counters.

```
var alive = 0;
var killed = 0;
```

5.2. Display the new "score" div:

```
var $score = $("#score");
$score.slideDown();
```

(More details: http://api.jquery.com/slideDown/)

5.3. Alter our map-clicking code:

a. We need to increment our *alive* counter when we create an orc
b. We need to decrement our *alive* counter when we kill an orc
c. We need to increment our *killed* counter when we kill an orc
d. We need to show the score whenever our counters change: we can do that by calling a showScore() function that we will create in the next step.

The dark red code is the new code:

```
$map.on("click", function(e) {
	var css = "width: 100px; position: absolute; ";
	css += "left: " + e.clientX + "px; top: " + e.clientY + "px";
	var $orc = $("<img src='orc.png' style='" + css + "'>");
	$game.append($orc);
	alive++;
	showScore();

	$orc.on("click", function() {
		$orc.remove();
		killed++;
		alive--;
		showScore();
	});
});
```

5.4. Now implement the showScore() function:

```
function showScore() {
	$score.html("Orcs killed: " + killed + "<p>Orcs still alive: " + alive);
}
```

Run your code!

Step 6

6.1. Let's make it interesting by animating the orcs!
Like in step 5.3, let's call a new function that we will implement in the next step: move($orc). Notice that we are passing our jQuery object (that holds "an orc") as an argument to this function:

```
$map.on("click", function(e) {
	var css = "width: 100px; position: absolute; ";
	css += "left: " + e.clientX + "px; top: " + e.clientY + "px";
	var $orc = $("<img src='orc.png' style='" + css + "'>");
	$game.append($orc);
	move($orc);
	alive++;
	showScore();

	$orc.on("click", function() {
		$orc.remove();
		killed++;
		alive--;
		showScore();
	});
});
```

6.2. Now implement the function:

```
function move($npc) {
      $npc.animate({left: "+=500"}, 3000);
}
```

The animate() method enables you to animate a numeric CSS property. In our case, we animate the "left" property, adding 500 pixels over the span of 3000 milliseconds (3 seconds). As a result, our orc moves 500 pixels to the right in 3 seconds. Try it!

Note: this code: `{left: "+=500"}` is an object we've just created. We haven't covered this in our class (not enough time in one semester!). The object has a property (left) and a value (+=500). We pass this object to the animate() function as an argument. The second argument is the duration: 3000 milliseconds.

(More details: http://api.jquery.com/animate/)

6.3. Now let's move the orc back:

```
function move($npc) {
      $npc.animate({left: "+=500"}, 3000);
      $npc.animate({left: "-=500"}, 3000);
}
```

6.4. Finally, let's use some raw JavaScript that we've learned earlier to make this even better: wrap our jQuery methods in a loop! Now the orc will keep moving!

```
function move($npc) {
      for (var i=0; i<100; i++) {
            $npc.animate({left: "+=500"}, 3000);
            $npc.animate({left: "-=500"}, 3000);
      }
}
```

I did not use a while loop because that won't work in the browser (that has to deal with how JavaScript is executed in the browser, which is not relevant to our current task). That's why I use a for loop instead.

6.5. Finally, let's add event handling to the "quit" button - so we can nuke all the orcs in one click:.

```
$("#quit").on("click", function(){
      $("img:animated").remove();
});
```

Here, we are using a jQuery filter: ":animated" that selects all img elements that are currently being animated.

Run your code!

Step 7

7.1. Finally, let's add a visual effect at the start of our game.

Instead of this code:

```
$messageBox.slideDown();
$score.slideDown();
$game.show();
$(".location").hide().fadeIn(2000);
```

Use this: (pay attention to the dark red code - it's the new code):

```
$(".location").hide();
$("#logo").fadeIn(5000).fadeOut(2000, function() {
      $game.fadeIn(1000, function() {
            $(".location").fadeIn(2000);
            $messageBox.slideDown();
            $score.slideDown();
      });
});
```

Here's what's going on here:

a. We hide all locations
b. We fade-in the logo (yay!!!)
c. We then fade out the logo
d. The fadeOut() method takes 2 arguments:
    1. the duration (2000 milliseconds), and
    2. a function!???
        i. that fades in the "game" div, taking 2 arguments:
            1. the duration (2000 milliseconds), and
            2. another function???
                a. that fades in the locations,
                b. slides down the message box and the score

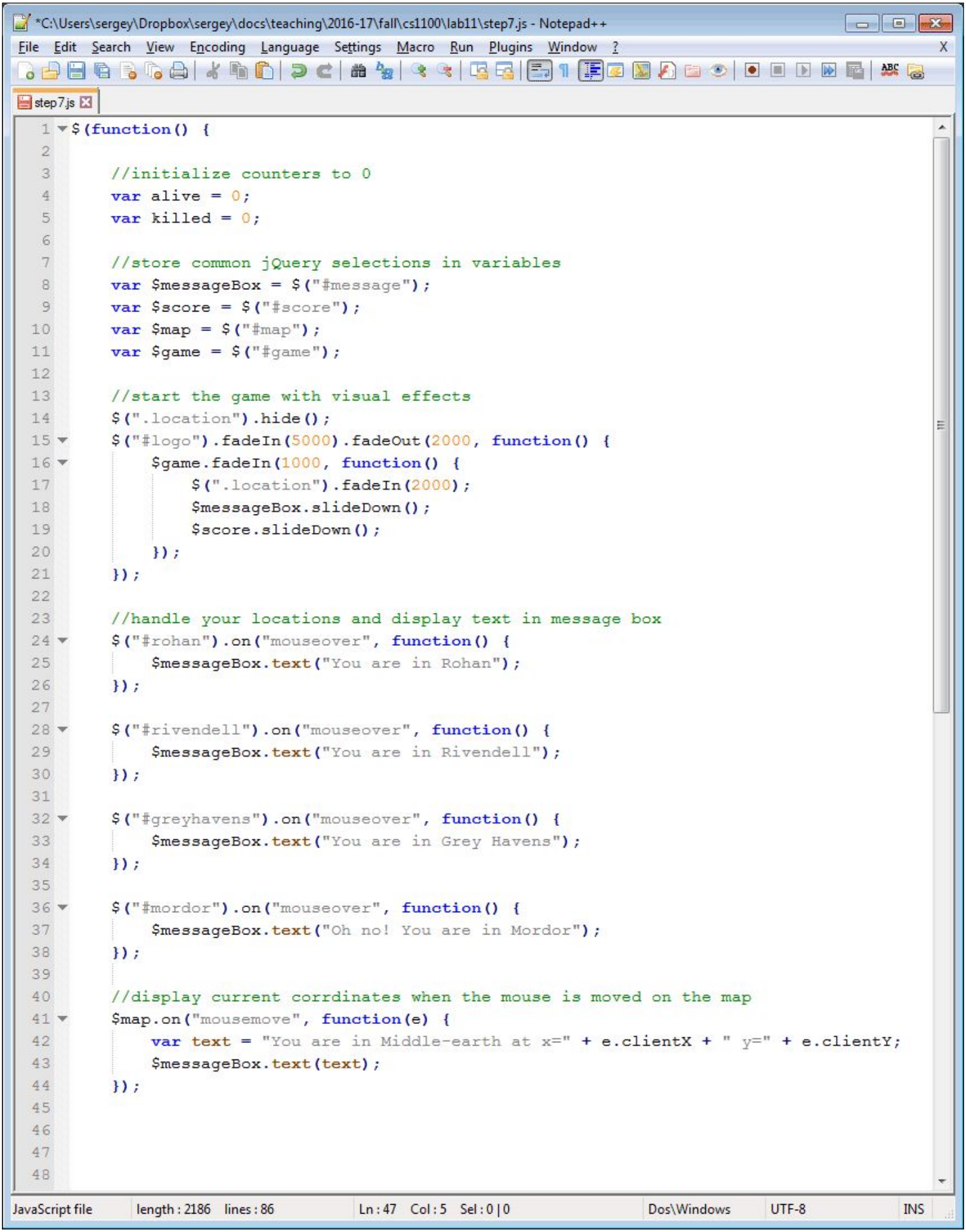
What is going on???
If we simply call animation A after animation B, they will execute at the same time. That's not what we want: we want animation B to start AFTER animation A is completed. To do this, we pass the function that takes care of animation B as an argument to A; that way we ensure that B is called only after A is completed. This can be challenging, so try to play with this code, change the values, and see what happens. You will feel more comfortable with practice, I promise!

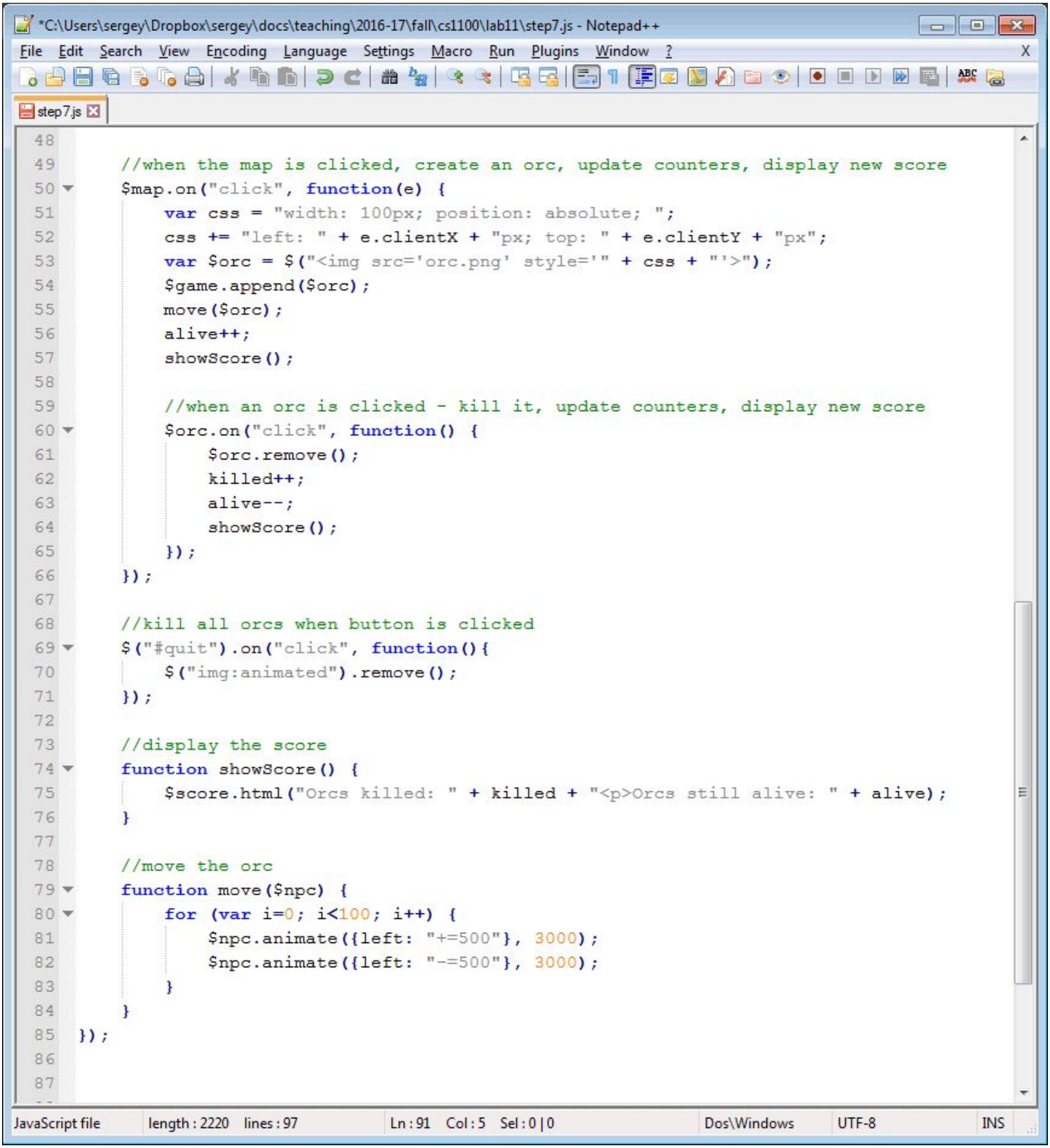
Run your code. You are done! Congratulations!

Submit your work

Submit **lab11.js** to eLearning:
https://bb9.uni.edu > log in with CatID > our course > Course Content > Labs > Lab 11

For Your Reference: This is what your code should look like in the end (see next page):


*C:\Users\sergey\Dropbox\sergey\docs\teaching\2016-17\fall\cs1100\lab11\step7.js - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
step7.js
$(function() {
//initialize counters to 0
var alive = 0;
var killed = 0;
//store common jQuery selections in variables
var $messageBox = $("#message");
var $score = $("#score");
var $map = $("#map");
var $game = $("#game");
//start the game with visual effects
$(".location").hide();
$("#logo").fadeIn(5000).fadeOut(2000, function() {
$game.fadeIn(1000, function() {
$(".location").fadeIn(2000);
$messageBox.slideDown();
$score.slideDown();
});
});
//handle your locations and display text in message box
$("#rohan").on("mouseover", function() {
$messageBox.text("You are in Rohan");
});
$("#rivendell").on("mouseover", function() {
$messageBox.text("You are in Rivendell");
});
$("#greyhavens").on("mouseover", function() {
$messageBox.text("You are in Grey Havens");
});
$("#mordor").on("mouseover", function() {
$messageBox.text("Oh no! You are in Mordor");
});
//display current corrdinates when the mouse is moved on the map
$map.on("mousemove", function(e) {
var text = "You are in Middle-earth at x=" + e.clientX + " y=" + e.clientY;
$messageBox.text(text);
});
JavaScript file length : 2186 lines : 86 Ln : 47 Col : 5 Sel : 0 | 0 Dos\Windows UTF-8 INS


*C:\Users\sergey\Dropbox\sergey\docs\teaching\2016-17\fall\cs1100\lab11\step7.js - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
step7.js
48
49 //when the map is clicked, create an orc, update counters, display new score
50 $map.on("click", function(e) {
51 var css = "width: 100px; position: absolute; ";
52 css += "left: " + e.clientX + "px; top: " + e.clientY + "px";
53 var $orc = $("<img src='orc.png' style='" + css + "'>");
54 $game.append($orc);
55 move($orc);
56 alive++;
57 showScore();
58
59 //when an orc is clicked - kill it, update counters, display new score
60 $orc.on("click", function() {
61 $orc.remove();
62 killed++;
63 alive--;
64 showScore();
65 });
66 });
67
68 //kill all orcs when button is clicked
69 $("#quit").on("click", function(){
70 $("img:animated").remove();
71 });
72
73 //display the score
74 function showScore() {
75 $score.html("Orcs killed: " + killed + "<p>Orcs still alive: " + alive);
76 }
77
78 //move the orc
79 function move($npc) {
80 for (var i=0; i<100; i++) {
81 $npc.animate({left: "+=500"}, 3000);
82 $npc.animate({left: "-=500"}, 3000);
83 }
84 }
85 });
86
87
JavaScript file length : 2220 lines : 97 Ln : 91 Col : 5 Sel : 0 | 0 Dos\Windows UTF-8 INS