SSHCI-5/433D Web Lab
Project 3:
Implementing a General State Machine in JavaScript
Due: Thursday, October 24, 2013 by 10:00 am

Goal

Your goal for this project will be to build general state-tracking system (a state machine) as described in SSUI lecture. The purpose of building a finite state machine is to simplify the task of building complex interactive elements such as resizable and draggable options. Because interactions with elements such as resizable icons occur over time, interactive elements must save their current state to reflect this input over time.

Files Provided

statemachinetest.js	JavaScript file that has one simple test of your state machine.
Statemachinetest.html	HTML file that goes with statemachinetest.js
statemachine-starter.js	Basic skeleton code you can optionally start from.

Project Overview

You will be creating a StateMachine object that models a state machine. Each state machine class can be attached to a DOM Element object. In other words, you can attach a state machine to a header, div, or any other DOM element. Your state machine will be described using a JavaScript object . The JavaScript object is described in more detail below. The JavaScript object will specify the states in your state machine, as well as transitions. **The start state of your state machine will be the first state in your state list**. When input occurs on any attached elements, the state machine will need to transition to a new state according to the input, and execute any functions necessary.

Your state machine should be able to take in any valid state machine specification and behave correctly according to the specification.

Detailed Specification

StateMachine Specification

For this project, a state machine will be described as a JavaScript object (in JavaScript Object Notation, or JSON). The state machine for this assignment will be described as a list of states. Each state has a list of outgoing transitions. For this assignment, the first state in the list of state machines is the start state. Transitions specify a target state (the state to transition to if the input matches), and the input to transition on. For this assignment transitions will also have a function (called an 'action') which represents the function to be executed when the transition is taken. Your action function should take a parameter specifying the event that just happened, as well as a

parameter specifying the element this state machine is attached to (so that your functions can make modifications to the elements themselves).

Below is an example of part of a state machine specification for a draggable element. The state machine description is also provided for you in statemachinetest.js:

```
var sampleDescription = {
      states: [
      {
            name: "start",
            transitions: [
                  {
                        input: "mouseDown",
                        action: record_down_location,
                        endState: "down"
                  }]
      },
      {
            name: "down",
            transitions: [
                  {
                        input: "mouseUp",
                        action: do_drop,
                        endState: "start"
                  },
                  {
                        input: "mouseMove",
                        action: move_icon,
                        endState: "down"
                  }
            ]
      }
]};
```

StateMachine Object

Constructor Parameters (in this order):

***description*:** Description of the state machine.

elementToAttach: DOM element to attach yourself to.

Methods:

updateState(input_event, input_type): Updates the current state given input, using the state machine table

Fields:

stateTable: The table the describes the finite state machine.

currentState: The current state of the state machine.

State Machine Input Events

Your state machine will need to translate raw input events into strings like “mouseDown”, “mouseUp” for interpretation by the state machine.Your statemachine must respond to the following events::

mouseDown: The mouse button was pressed down over this object.
mouseUp: The mouse button was released over the object.
click: The object was clicked.
mouseMove: The mouse has moved over the object.
mouseIn: The mouse has entered the object.
mouseOut: The mouse has left the object.
keyPress: A key was pressed.
timerTick30Ms: A timer that goes off every 30 milliseconds has gone off (useful for animations)

Testing Your State Machine

You are given one simple state machine to test out your code on. This state machine makes the div "myDiv" draggable. Additionally, when the div is pressed it should turn blue. Finally, the state machine logs each of its transitions to the console for debugging purposes.

You should write at least 5 more tests to test out whether your code works. These tests can be simple, such as the draggable div in the testing code, or more interesting, such as an implementation of a simple fighting game (key combinations could let your characters do cool moves). Another idea for a test would be to do a complex and interesting animation sequence, as you can track what part of the sequence you're in using a state machine. Note that your sample doesn't have to be one giant state machine. You can have several state machines in one sample. **In fact, one of your tests must have more than one state machine** to test how having multiple state machines in 1 HTML file affects your result.

Additionally, you can make your HTML page as interesting as you'd like (for example, you can add an interesting background to set the scene for a fighting game). You are free to use jQuery to make animations or to simplify development, however be warned, this might make your life more complicated!

The sample test code only responds to mouseDown, mouseUp, mouseMove, and mouseOut events. Collectively, your tests should test each of the input events described above. This doesn't mean that you need to test every input event type *per test*, but means that *across all tests,* you must cover every input event type.

Additionally, **each of your tests must implement at least 3 actions** (functions that do something on a transition), but for more interesting tests, you will probably want to have more than just 3 actions.

Bells and Whistles

Completing the above requirements and having well-documented code with no errors will get you 45 out of 50 points, which is an A. You must complete one of the below 'bells and whistles' to get full points. You can complete as many of these as you want, but you will get no more than 10 points total. Different bonuses are worth different numbers of points, based on difficulty. You may receive up to 10 points for bells and whistles, bringing you up to up to 5 extra credit points. I hope these are fun for you!

1. **Make an interesting game as one of your tests (up to 10 points):** State machines are particularly useful for making games where you need to use keyboard combinations (think street fighter), or for making puzzles where you need to a set of things in the correct order to unlock a door. Make one of your tests an interesting game that uses state machines to track state for up to 10 points.
2. **Write a visualization for your FSMs (10 points):** Make a visualization of all of your states & transitions, including highlighting the active state. You may want to consider using the drawing library you built in P2.

Show and Tell

The course instructors will be picking the most interesting state machine tests (i.e. the most interesting state machines you created) and showing them to the class. They may also be put on a public website. Please let me know if you do not want your solution shown in the README file in your assignment.

Turning Your Program In

This project has the same procedure as Projects 0, 1, and 2. Make sure to include the following files:

statemachine.js	Your state machine implementation
statemachine-test1.html and .js	Your first state machine test.
statemachine-test2.html and .js	Your second state machine test.
statemachine-test3.html and .js	Your third state machine test.
statemachine-test4.html and .js	Your fourth state machine test.
statemachine-test5.html and .js	Your fifth state machine test.
README.txt	Describe your project as in previous projects.

If you do any bells and whistles, also include the testing files for those bells and whistles. Make sure your README file includes any extra documentation about any extra things you did. Also, include any questions or difficulties you had. If something doesn't work, please also include this in the README file.

Grading

Turn in is correct, code has no syntax errors	5 pts
Formatting, Comments and Coding style	5 pts
Correct implementation of StateMachine class	15 pts
Elegance of implementation of StateMachine class	5 pts
At least one test covers multiple state machines	5 pts
Tests thoroughly test all components of State Machine.	10 pts.
Bells and whistles	Up to 10 pts