SER421 Fall 2017 Online Lab4: SPA
Assigned 11/2/17, due 11/13/17 at 11:59pm via online submission to Blackboard

Objectives:
1. Demonstrate proficiency in constructing a self-contained (single-page) HTML/Javascript application (SPA) using HTML DOM manipulation

Overview:
For this lab you will develop a self-contained HTML5/Javascript application. The intent of the lab is to ensure you are comfortable with the DOM representation within the browser, as it will serve as a basis for much of what we do in the 2nd half of this session.

You may complete and submit this lab in pairs!

Task: Construct a self-contained Javascript Application
For this task you will create a self-contained, or *single page application (SPA)*, for a simplified version of the board game *Clue*. Clue is a popular board game where guests are invited to a dinner party, only to have one guest murdered by a secret guest, in a secret room, with a secret weapon. It is also the subject of an awful movie (www.imdb.com/title/tt0088930) from 1985. The real game of Clue is a multiplayer game (en.wikipedia.org/wiki/Cluedo). The environment consists of 6 guests, 9 rooms, and 6 weapons. There are assorted rules for moving around the board and guessing suspects and circumstances in a process of elimination until a player declares "whodunit", in what room, and with what weapon.

You will construct a drastically simplified version of the game with two players, you and the computer. The suspects, weapons, and rooms in your environment will be pre-set as separate global arrays. You and the computer will be given an equal number of cards with 3 cards held in reserve as the triplet with the secret to guess (you may assume there are 21 cards as described above, so when the secret triplet is removed, there are 18, or 9 per user/computer). You and the computer will take turns guessing triples until one of you arrives at the solution (matches the secret).

Functional requirements:
1. Display the full set of suspects, rooms, and weapons at the top of an HTML page for the user to see.
2. Underneath that, provide a textbox prompting for the user's name with a Submit button. When the user's enters her/his name, replace the form with a dynamic message saying "Welcome <name>".
3. Implement a shuffle function that:
    a. First, randomly selects the secret triplet (one room, one suspect, one weapon)
    b. Then, randomly distributes ½ of the remaining cards to the user and ½ to the computer (9 each)
4. Display the set of "cards" the human user "holds in her/his hand".
5. Display an HTML form that allows you to select 1 suspect, 1 weapon, and 1 room. I suggest you use a choice box for this but in fact it does not matter what you use as long as it is an HTML form. The available options in the selections must NOT include those cards that the uer already has in her/his hand.
6. The submit button for your form should cause Javascript to read the form values, and compare them to the secret.
    a. If the guess matches the secret, dynamically display a "win" message
    b. If the guess does not match the secret, dynamically display a message stating 1) the guess did not match the secret, and 2) Reveal ONE AND ONLY ONE incorrect component of your guess.
    c. Provide a "Continue" button that either resets the game to the beginning (if the user won), or allows the Computer to move (next step).
7. When the user's (incorrect) move is complete, have the Computer make a guess. A simple random guess will do (or any algorithm where the Computer does not repeat a guess), though the Computer should know not to guess cards that it holds. Like #4, display a message indicating whether the Computer's guess was correct or not, and a Continue button at the end. Like before, you should display a card that the user holds in her/his hand, similar to 6.b.2.
8. Provide a button (outside the form) named "Show History" that dynamically shows the guesses you and the Computer have made so far in the current game. When shown, the "Show History" button should become "Hide History" (you are essentially creating a toggle button). This history should reset each time the browser closes.

9. Provide a button named “Show Record” that when pressed, dynamically displays the won-loss record for the Computer, and a history of who the Computer played, the date, and the outcome. This record should not reset when the browser closes, or if Cookies are cleared.
10. If the user hits the browser's refresh button, the SPA should reload. You will have to take care to ensure that the state of the game is reset including the "Show History" and "Show Record" features (this is a clear hint!).

You may navigate to http://www.public.asu.edu/~kgary/ser421/lab4/sample1.html to see a mock version of the game.

Please submit your solution as lab4.html and lab4.js. No third-party dependencies are required or allowed for this lab.

Nonfunctional Requirements:
The intent of this task is to familiarize yourself with Javascript, and some common straightforward techniques, like responding to UI events, DOM manipulation, reading forms, and storage APIs. Therefore:

1. Do not load a new file using something like window.location.href assignment. The application is to be a SPA.
2. Do not use alerts. “Dynamic display” in the functional requirements means you are to dynamically manipulate the DOM to substitute new content as needed.
3. You may assume the data will be initialized as global variable arrays named, aptly enough, *suspects*, *weapons*, and *rooms*. While the Clue game uses 21 cards, your game should work for any multiple of 2n+3, where is at least 2 (that is 7, 9, 11, 13, etc.). So do not hardcode the number of cards.
4. Functional requirements #8 and #9 should be implemented using Session/Local storage (you decide which to use for what requirement). Functional Requirement #10 is there so you have to think about how to maintain UI state consistency in the face of a changing game state.

Grading hints – I suggest you decompose the functional requirements and complete in increments. I will award partial credit, so you are welcome to submit 2 files, one representing your latest working version and one your latest non-working version. I view requirements 1-5 as straightforward, 6 and 7 a little more complex, and 8 and 9 as related. Again, have a README.txt for anything we should know.

Extra Credit 1: (25 points)
Clue (like most board games), is much more interesting if played visually. Modify the text-based Clue game of your requirements to be a visual game using drag-and-drop of images, replicating the behavior shown in the video at http://www.public.asu.edu/~kgary/ser421/clue/gameplay.mp4. The media will be provided to you on Blackboard. Name your HTML file lab4_ec.html, and your Javascript file lab4_ec.js. We will give you some CSS to help too.

Submission:

You may implement your solution **with a single partner if you choose**, though you are not required to partner if you do not want to. Submit via a zipfile named <asurite>_lab4.zip on Blackboard by the due date. Make sure the name of the HTML file for us to run your SPA is called lab4.html with Javascript lab4.js. You may add a README.txt in the zipfile if there is anything you want us to take into consideration. If you do partner with someone make sure both names are in the README file, and only one person needs to submit. There will be no extensions for this lab!