EECS 1012: LAB 08 – Code Breaker

A. IMPORTANT REMINDERS

1) Lab8 is due on **Friday (Apr 7)** at 9pm. No late submission will be accepted.
2) You are welcome to attend the lab sessions on next Monday (Mar 27) and Tuesday. TAs and instructor will be available to help you. The location is WSC105 (Mon) and WSC106 (Tue). Attendance is optional. You are encouraged to come to your lab session, but you don't have to if other time works better for you. (Later for lab tests, you need to come to your official lab session.)
3) Feel free to signal a TA for help if you stuck on any of the steps below. Yet, note that TAs would need to help other students too.
4) You can submit your lab work any time before the specified deadline.

B. IMPORTANT PRE-LAB WORKS YOU NEED TO DO BEFORE GOING TO THE LAB

1) Download this lab files and read them carefully to the end.
2) If you are not familiar with the Code Breaker board game, visit https://en.wikipedia.org/wiki/Mastermind_(board_game). Note that the one we make in this lab has 5 code pegs (not 4).
3) You should have a good understanding of
   - Document Object Model https://www.w3schools.com/js/js_htmldom.asp , in particular **createElement**, **append**, etc. https://www.w3schools.com/jsref/dom_obj_document.asp
   - jQuery https://www.w3schools.com/jquery/
   - Revisit Slides and recordings for this week and identify the jQuery commands and review their meaning. We highly encourage you—prior to go to labs this week— review the demo jQuery code posted on eClass, and, to do two sets of exercises on jQuery selectors and events available in w3schools, starting here: https://www.w3schools.com/jquery/exercise_jq.asp?filename=exercise_jq_selectors1
   - JSON https://www.w3schools.com/js/js_json_intro.asp , in particular **stringify**, and **parse** methods. https://www.w3schools.com/js/js_json_stringify.asp and https://www.w3schools.com/js/js_json_parse.asp
4) Understanding AJAX **request** and **response** is also an asset, even though we are not going to use it directly. https://www.w3schools.com/js/js_ajax_http_send.asp and https://www.w3schools.com/js/js_ajax_http_response.asp . This topic will be clearer by end of this lab when you address about the server side too.
5) Asynchronous JavaScript and XML (AJAX) is an important concept—in JavaScript—for sending requests to servers asynchronously. AJAX syntax is well simplified in JQuery's **get** and **post** methods. You may want to revisit these methods before doing this lab.
6) You should have a good understanding of **JSON**, **stringify** and **parse** methods.
   - Encourage you to revisit it here https://www.w3schools.com/js/js_json_intro.asp , and make sure you have a clear idea about what **stringify** and **parse** methods are for (https://www.w3schools.com/js/js_json_stringify.asp and https://www.w3schools.com/js/js_json_parse.asp).
7) In the server side (Task 2), we use **node.js** (the JavaScript server environment, https://www.w3schools.com/nodejs/) together with one of its popular web frameworks, called **Express**. What are in lecture notes are sufficient for a starting point of **Express**. But, if you want to learn more, visit here https://expressjs.com/en/5x/api.html#app and here https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction .
8) You need to install **Node.js** and **Express.js** on your computer before doing Task2 of this lab. Instructions are given at the end of this manual.

C. GOALS/OUTCOMES FOR LAB

1) To become familiar with JQuery and JSON
2) To work more with DOM

3) To become familiar with **Node.js** and **Express.js**
4) To complete the development of a fully-fledge web application, getting preparedness for your ongoing project

D. TASKS

1) TASK 1: Client-Side of the Code Breaker Game.
2) TASK2: Server-Side of the Code Breaker Game

E. SUBMISSION

eClass submission. More information can be found at the end of this document.

F. Task 1 client side

Use html, CSS, and JS to design the client side of the code-breaker game. The server is up running on the EECS department's machine **indigo.eecs.yorku.ca** at port 3000. We have provided you with a starter code, which connects to the server. You should:

1) Open **code_breakerV0.html** and save it as (client side) **code_breaker.html**. You should read the code and comments in the html file. The file creates the initial game board. Next, following the comments, make changes such that your html file becomes similar to **code_breakerV1.html**, eventually. Note that the html file should be shortened from 150 lines to 28 lines.

2) Next, open **code_breaker_clientV0.js** and save it as **code_breaker_client.js**. In the **createGameBoard** function of **code_breaker.js**, you should read the comments and write the code for all 15 lines specified by "**//…**". These lines dynamically build the html tags that you just deleted from your html file: you are creating those tags via your JS file at run time (dynamically). Note that you can use jQuery notation, or, use regular JS notation. Also note that you should not change any code outside of the **createGameboard** function.

If you make the above changes properly, your code_breaker should work fine, that means you can start playing the game by opening your **code_breaker.html** in Firefox.

Note: we make sure that our **code_breaker_server** is up on **indigo.eecs.yorku.ca** from Mar 23 until Apr 2. You are expected to finish and test your client code by then. In task 2, you will complete your own server side code such that you are able to run the server on your computer any time. (This is also what is expected for your project – have a local server file up running and have a client side code talking to the server.)

G. Task 2 Server side -- **do this task only after you have completed Task1, and your client side code works correctly with the remote sever at indigo.eecs.yorku.ca.**

In this task, you complete the **server-side JS file**. What do you need to do in this task is the following 4 steps:

i) Revisit details of the requests that are sent from the client-side: one is in **initGameBoard()** and the other is in **processAttempt()** function. Study the given code in the **code_breaker_client.js**, and make sure you understand the two functions.

ii) Also, revisit slides of client side code. Note that when the request is responded by the server, a callback function is called automatically. We named the callback function *response*. The details of the response function were illustrated in slides and can be found in the **code_breaker_client.js** we already provided you with.

iii) Copy **code_breaker_serverV0.js** to a new file named **code_breaker_server.js** and follow the comments to complete the code. In particular, there are ten /*TODO…*/ comments that you should replace with your code. More detailed instructions can be found in the file. Below are some more hints on each of these concepts:

/*TODO 1 …

Note that the **req.query['data']** will return the part of the request that is specified by **data**, which is in JSON format. If we assign that to a variable name z, we can access components of that with **z['name'], z['action'], z['attempt_code']**, and **z['current_attempt_id'].** This is valid because in JavaScript, we can treat an object as an *associative array*, where the index is 'name', 'action','attempt_code' etc.

/*TODO 2 …

Once we have prepared a response in a string format, we can send the response back to the requester (client) using method **send**() where we pass the string as a parameter to this method.

/*TODO 3-8 …

One of best hints on how to do these 6 TODOs is to see the Slide, where the associative array **response[]** has been used in the client side to figure out what data the server has sent back in its response. Just recall that objects in JavaScript could be defined as:

```
{
    property1: value1,
    property2: value2,
    property3: value3,
    …
}
```

/* TODO 9 …

If you look at the Port # that requests are sent to from the client side in the **code_breaker_client.js** file, this becomes trivial.

/*TODO 10 …

Here, you just want to declare an empty array. The array will be populated by 5 unique id of marbles in the following lines.

iv) Finally, follow the instructions at the end of this manual to run the server locally on your machine and play the game on your device any time. Note: you need to change the url in the client side JS code from "http://indigo.eecs.yorku.ca:3000/post" to "**http://localhost:3000/post**", and then load the html.

Note: since you are running the server on your local computer, you can see the output by the server code, examining how the messages are transferred between the client and server. E.g., each attempt is transferred to the server as an array of peg ids, like [5, 4, 3, 6, 1]. And the server responds back with "num_match", "num_containing" etc.
Also, you can 'cheat' by looking at the generated code for you (displayed by the server), then enter accordingly to win immediately 😊.

H. AFTER-LAB TASKS (THIS PART WILL NOT BE GRADED)

In order to review what you have learned in this lab as well as expanding your skills further, we recommend the following questions and extra practices:

1) This lab is a great source for your on-going project. It is also a great source of learning. Read all comments carefully and make sure you have a clear understanding of the code—line by line. Many of lines of this code are just advanced programming some of which somewhat above the expectations of a typical CS1. Yet, we are sure many of you have a great thirst to learn more. So, this code should serve you for that purpose.

2) An interesting way to enhance this project is as follows. Add a button "Hint", such that every time the user clicks on it, the server reveals one of the five pegs, in any order you wish. The server should not reveal more than 3 pegs. In order to implement this component, you need to add a new action to your requests. So far, you had two actions: "generateCode" and "evaluate". Let's call the third one, "hintMe".

3) Make sure you understand objects in JavaScript can be treated as associative arrays and vice versa. See the following example:

```
var student = {     firstName : "Anna",
                    lastName : "Smith",
                    major : "CS",
                    age : 23 };

var x = student.lastName;          // is the same as  var x = student["lastName"];
```

See this page for further information on arrays and objects: https://www.w3schools.com/js/js_arrays.asp

Please feel free to discuss any of these questions in the course forum or see the TAs and/or Instructors for help.

I. SUBMISSIONS

eClass submission

You should have a folder named "**Lab08**", and have **all** of your HTML and JS files; as well as the installed folder **node_modules** and some **package??.json** files.

Compress the whole folder and upload the zip (or tar) file to eClass. (Make sure to include the **node_moduels** folder and **.json** files)

Installations (needed for Task 2)

Step 1: install node.js (You should have done this for lab7. If not, see instructions in lab7 manual)

Step 2: install express. In the terminal, navigate to your lab8 directory (using *cd* command). Then issue

npm install express

Step 3: run nodejs server (for Task 2). In the terminal, navigate to your lab8 directory, and issue

node code_breaker_server.js

If the server is up successfully, in the terminal you should see output *"sever is running! (listen on port 3000)"*

Step 4: open the html file *code_breaker_client.html* in browser to play the game

To stop the server, type *ctrl + c* in the terminal