Unit 5 Part Two Diary Entry

Matthew Zarowny

ID: 3484796

Athabasca University, 2021

Link to Site:

http://student.athabascau.ca/~mattewza/Unit5/html/home.html

Work Done

In this unit, I implemented the 3 ideas I outlined in Unit 5 Part One. I detail the work done for each of these ideas in the following subsections:

Idea 1 – Accordion for FAQ Page

The accordion displays all of the questions in a list. Then, when a user clicks on a question, the answer to that question will slide out from the question and can then be easily read. The code functions such that only one answer is shown at a time, i.e. if a user clicks one question and then another, the answer to the previous question will become hidden and the answer to the new question will be displayed.

In order to implement the accordion functionality, I had to modify my existing HTML code for the FAQ page as well as write a new CSS and JS file.

I modified the HTML file by getting rid of older tags and replacing them with tags and <button> tags that could be used in tandem with my new CSS and JS file. Questions in the FAQ now have a <button class = “question”> tag and the answers have a <div class=”answer” tag>

The JS file (accordion.js) consists of a few global variables and a function. The code first gets the question elements from their class name. Then, a for loop adds an event handler for each question element that will execute a function once the button is clicked.

The function employs DOM in order to associate the question element with the answer element (the answer class is the next sibling to the question class). Then, the question elements have the class “active” added to them and the maxHeight of the answer class is set to its scrollHeight. In combination with the CSS code, this allows the answers to “slide” out from the questions. There is also an if else chain that is used to determine if there are any open answers that need to be closed to accommodate the newly opened answer. For more details, please see the commented code in “accordion.js”

The CSS code is very similar to the CSS file used in Unit 4 for the “read-more” buttons. It is called “accordion.css”.

Idea 2 – Dynamic Table Filtering for the Leaderboard Page

The dynamic table filtering of the leaderboards allows users to hit a button that will filter/de-filter such that only leaderboard entries with media links are shown. It also contains an input bar where the user can type text to filter the leaderboard based on player names. This filtering is not case sensitive and only a section of the player name needs to be typed by the user, exact matches are not necessary.

In order to implement the filtering functionality, I had to slightly modify my existing HTML code for the leaderboard page by adding a new that served as a container for the table. I also created two brand new files: “tableFilter.js” and “tableFilter.css”.

The new JS file is composed of a couple global variables and three functions. One function filters the table’s rows by name based on the input of the user. It is called every time there is an upkey event in the input element. The second function filters the table’s rows by if they have a media link or not. It is called every time there is a click event on the filtering button. The third function I realized I had to write later. It allows for both types of filtering to occur without bugs at the same time. The code uses DOM to access the relevant elements from the table and then uses if else chains to determine if rows should be displayed or not. Please see the commented code for further details on how I implemented this.

The new CSS file makes the table look a lot nicer and also highlights the row that the user’s mouse hovers over for easier readability.

Idea 3 – Slideshow for the Setup Page

The slideshow on the setup page allows users to cycle through the different methods of setting up the game. Each method of setting up the game is displayed in a box at the center of the screen. When the user hits the arrows on the left and right side of this box, the next method and relevant information will be displayed in the box. There are also circular indicators at the bottom of the box to indicate to the user where they are in the chain of slides and these indicators can also be clicked to make the box display a specific method.

In order to implement the slideshow functionality, I had to slightly modify my existing HTML code for the setup page by adding new tags that served as containers for the slides as well as adding new and <a> tags for the circular indicators and arrows respectively. I also created two brand new files: “slideshow.js” and “slideshow.css”.

The new JS file is composed of a few global variables and three functions. The global variables store the slide elements and the slide that is currently being shown on the screen. The first function is called when the user clicks one of the arrow keys. It increments/decrements the index of the active slide depending on if the left or right arrow was pressed before calling the third function to change the slide to the new slide. The second function is called when the user clicks one of the circular indicators and it then sets index of the active slide to be the index of

the clicked indicator. This function then calls the third function to display the new slide. The third function sets the display of every slide to be none except for the new active slide, which is set as the active slide. For more detailed information on the work I did, please see my commented code.

The CSS file puts the two arrows on the left and right of the slideshow box. The arrows themselves are text characters. When the user hovers their mouse over an arrow, a box surrounding the arrow will fade in. The file also places the three circular indicator buttons below the slideshow box. They take on a different colour when they are displaying the active slide or if they are hovered over by the user's mouse.

Rationale Relating to Personas and Scenarios

I will outline the design rationale for each of three ideas individually in the following subsections.

Idea 1 – Accordion for FAQ Page

The accordion functionality makes the FAQ page overall a lot easier to user. It handles scenarios where a user wants to find a quick answer to something very well

The accordion benefits personas like Luke by providing them with a long list of questions in a small space. Personas like Luke are somewhat serious users of the site and want to get information as quickly as possible. The accordion facilitates this by allowing one to easily find the answer to the specific question they have without scrolling through a massive page of text.

The accordion also benefits the more casual personas like Sally. If these types of users saw a massive page of text on the FAQ, it may cause them to feel overwhelmed by information and text. The accordion prevents this from happening by obfuscating all of the text and presenting the questions in a compact space.

Idea 2 – Dynamic Table Filtering for the Leaderboard Page

Table filtering will be values by personas like Luke. Since these types of personas are more serious users, they may want to look for specific names in the leaderboard, or only look for the entries in the leaderboard that have video links that they can watch. The filtering allows these types of personas to easily get through these scenarios efficiently by providing them with an alternative to scrolling through a massive leaderboard of names.

I think that the design of the table has also improved, and this may benefit personas like Tom. It appears a lot more professional now, and a scouring knowledge-seeker type persona like Tom may appreciate this aesthetic.

Idea 3 – Slideshow for the Setup Page

The slideshow serves to benefit personas like Tom and Sally and serves to meet scenarios for when users are coming to the site for the first time. In this scenario, the user will want to setup the game and try it to see what the fuss is all about, so the setup page is likely their first

destination (note, that the home page also has a link to the setup page that serves to funnel new users directly to the setup page). Therefore, it is important that the setup page makes a good first impression on these types of users. Before, it was more like a wall of text and could be perceived as somewhat overwhelming and intimidating by more casual personas. Now, only a small portion of the screen is taken up by an aesthetically pleasing slideshow box. Users can interact with the slideshow box in order to find the key information that they need in order to setup the game.

Learning Outcomes

I have met the learning outcomes for this unit. I wrote well structured, well commented and easily maintainable JavaScript code. I used a very wide range of programming constructs and design methods to make modular code, including:

- Functions
- Global and local variables
- Arrays
- Passed parameters
- Classes
- Objects

I follow good coding practices and documented my code in a very understandable fashion. I also pay heed to correct indentation and camelCase practices. The code is accessible to my persona types and has been successfully debugged.

What Went Well, What Didn't

Overall, the coding process went fairly well. The pseudocode I laid out in part one served as an excellent guide for how I could go about structuring and writing my code. Unit 4 had already put me back into the JavaScript state of mind, so I was aware of what syntax to use in most places. Whenever I wasn't sure of what the correct syntax was required to achieve the functionality I wanted, I was able to quickly use web documentation in order to find it.

I always expect that some amount of debugging will be involved when composing code for any purpose. I did have to spend some time debugging all three of my JavaScript files. That being said, my debugging for the table filtering code did not go as well as I would have like it to. The bug that took me the longest time to remove was that the link filtering and name filtering did not seem to work in tandem with each other. For example, if I were to filter by name, then filter by link, and then filter by name again, rows in the table that should have been filtered out by link would appear anyways. I was able to eventually solve this and other similar problems by creating a third function that I did not outline in my pseudocode and also needed to add some Boolean control variables. I talk about this a bit more in the following section.

What I Would Change if I Did it Again

I would definitely change how I implemented the filtering in the table. This code was the code I spent debugging the longest and my solution ended up not being as modular as it could have been. For example, if I wanted to add another method of filtering the table, I would have to add

several new functions and control variables in order to achieve the correct functionality. If I were to code table filtering again, I would spend some more time coming up with a different solution and coding structure.

The Most Surprising and Useful Thing I Learned

Sometimes the simple things are most surprising. I learned that the .indexOf() method existed in JavaScript. I had no idea that this method existed, and while it's not a ground-breaking realization, learning of this method saved me some time while coding my table filtering code. I did not have to write extra loops to cycle through the characters of strings and then compare that way, I could just use this method. Again, a simple thing, but surprising and useful nonetheless. Since I'm so used to coding in C, I guess it's just surprising to me in general to learn that there seems to be a JavaScript function for everything.

How My Previous Experience Impacted My Work

My previous experience with JavaScript and coding in general definitely expedited the design process. I was able to write well-structured code right away and was able to use effective debugging strategies that I had previously learned.