Group Number : 116

- Drew Underwood
- Rajiv Bhoopala

Server setup through XAMPP.

In order to run our code, these steps must be taken.

- Run your XAMPP (or other viable server) and store all attached .php and .html files into your root directory.
- Create new database “portfolio” was all we named ours
    - “easyleaderboard” table
        - TimeCompleted INT(10)
        - UserName VARCHAR(255)
    - “intermediateleaderboard” table
        - TimeCompleted INT(10)
        - UserName VARCHAR(255)
    - “expertleaderboard” table
        - TimeCompleted INT(10)
        - UserName VARCHAR(255)
    - “users” table
        - UserName VARCHAR(255)
        - FirstName VARCHAR(255)
        - LastName VARCHAR(255)
        - Email VARCHAR(255)
        - Password VARCHAR(255)
- Initial steps for running the project require navigating to “Signup.html”
- Create a new user with a specified password, email, first and last name
- Once created, you will be directed to “login.html” where you can login with your recently created user
- Once logged in you will have three options presented to you, as well as all three leaderboards
    - EASY
    - INTERMEDIATE
    - EXPERT
- Each button is a difficulty selection, once clicked will begin the game
- If the game is lost, you will have the option to restart under any difficulty
- If the game is won, you will have your time entered into the leaderboard appropriate for the difficulty completed.
- Once finished, the option to logout is presented in the top right corner.

Check List (Performance)

- Signup Page
    - ☐ User creation relies on RegEx matching for:
    - ☐ Username (Alphanumeric)
    - ☐ Password (Matches confirm password field)
    - ☐ Email (Matches any possible email format)
    - ☐ First and Last Name (Alphabetical)
    - ☐ New User is added to “users” table, with encrypted password (md5)
- Login Page
    - ☐ Match user to database
    - ☐ Match password to user from database
    - ☐ If successful match, navigate to game page
    - ☐ Leaderboard is updated successfully upon completing a game after logout
- Title Page (titlePage was the designated gameplay page)
    - ☐ Difficulty selection, starts game without refresh or redirect
    - ☐ Once game is won (or lost) difficulty selection is redisplayed without refresh
    - ☐ User name is displayed in top right corner at all times
    - ☐ Logout button redirects to login page, with updated scores

Check List (Code)

- Signup.html
    - ☐ Click event handler takes in form data
    - ☐ Form data submitted through ajax call to Signup.php
    - ☐ Upon successful ajax call, if data is valid, redirect to login
    - ☐ Upon failure to meet data specifications, error message displays
- Signup.php
    - ☐ Establish connection to database
    - ☐ Retrieve specified user, match entered password
    - ☐ RegEx used to validate user input
- Login.html & Login.php
    - ☐ Same checkboxes for Signup.html & Signup.php
- titlePage.html
    - ☐ Initialize game: Constructs game based on difficulty
    - ☐ Display game: Constructs table for game, assigns ID based on position
    - ☐ Minesweeper class:
        - ☐ Builds grid with randomly assigned mines
        - ☐ Initializes counts of all cells, (-1) for mines, and for all other cells a number is given for how many mines are adjacent
        - ☐ Clear region, a recursive algorithm where if a cell with count ‘0’ is selected, all other zeros and their neighboring cells will be revealed
    - ☐ Cell Class:
        - ☐ Constructs a cell which allows for easy manipulation of what is clicked inside a grid for the game

Minesweeper

By: Drew Underwood and Rajiv Bhoopala

Group: 116

For our Portfolio, we have recreated the game Minesweeper in a simple table implementation.

Table of contents:

We decided for our project to redo the game Minesweeper. This game is simple and fun but, when being designed to fit into a web browser and function properly we ran into a couple of road blocks. The biggest issues faced were being able to model the game in a way that made sense, simple putting together a table and trying to have all of the act accordingly in a modeled array was not an easy task. Thus, a different method of approach was selected where we revisited our old habits of object oriented programming and tackled the algorithms required to run Minesweeper with greater ease. The game of Minesweeper has instructions found easily on google or can be derived just by playing the game. In short, a grid is displayed which has a number of hidden mines. The players objective is to avoid setting off any of these mines and will be shown numbers whenever they successfully find a block which is not a mine. Each number displayed gives the player an idea of how many mines are adjacent to that particular cell. Once all numbered cells which are not mines have been revealed, the player has won and their time is stored in the database for bragging rights. The tougher parts of this project revolved around keeping every aspect of the game in real time without refreshing the page as well as the recursive clear region algorithm.

New and Complex things:

This project required us to tackle four main things which were new and rather difficult to handle.

1. Clear Region recursive algorithm

   The Clear Region algorithm is a way of speeding up the game and keeping it interesting without having to go through the repetitious task of clearing out all the zeroes manually (Zeros being cells with no adjacent mines). Implementing this algorithm requires adaptive for loops as our cells are all stored within a two dimensional array. Since there is always a requirement to check all neighbors surrounding the current zero, boundaries must be set in order to avoid going outside of the game. Overcoming this required comparisons of maximum and/or minimum boundaries with the current position of the zero and then adding/subtracting from that position. Once boundaries have been established, there is to be a recursive call within the for loop, which will continue to call itself on any cells identified as zeros. As each zero is identified, it is revealed to the player, as well as all nearby non-zero/non-mine cells.

2. Javascript class system and strict object oriented practice

   Minesweeper prompted the use of a table to initialize all cells in a fashion that was accessible and easy to manipulate. However, when we began our project we quickly found that assigning individual cells to a grid and relying on our knowledge of each cell only in respect to their immediate neighbors was faulty, inconsistent, and resulted in a fairly unorganized structure. Thus, we resolved to create a Minesweeper and Cell class which has only been available under Javascript for a few years now. Assigning a grid to a Minesweeper object, and giving each grid location a Cell with all the necessary attributes to keep track of, allowed for systematic tracking and manipulation of all cells.

3. MySQL database organization and Leaderboard design

   The MySQL database design was similar to that of our Lab6 instruction. We built on this design by providing an ordering to the time it takes each player to complete their game.

4. Thorough understanding of asynchronous callbacks

   With running the game, each click requires updating the grid in real time. This was achieved primarily through JQUERY, but once the game is completed, or whenever we wished to verify information from Signup.html or login.html there is a requirement to have code execute upon successful callback from the server.

Blooms Taxonomy

Analyzing:

When beginning this project, we understood well enough how to play the game of Minesweeper. However, there was a requirement to be able to break the game into parts which we could understand and how each of these parts could be written down into a programming language we were familiar with. The first steps taken in analyzing our problem was what parts of this game do we want to break up. We viewed the game quite literally from the perspective as a grid. Knowing that this game was so structured, we were able to model multiple solutions to building the game at least with a visual skeleton that modeled our end result.

Once we were able to conjure a visual structure for our game, we had to start asking questions of how we wanted it to work. Did we want to have users which were created and stored, or did we simply want a game without any competitive nature? Should there be a leaderboard if decided to allow a competitive atmosphere? What information do we want from our users should we decide to have an account made under each user? Each of these questions brought us a clearer visual of what our end result needed to be.

Once the structure of the game had been thought on, we had to start configuring a plan on what to actually include in our code that would perform the operations of the game. Instantiating the game, each click having the proper effect on the game, how to handle a win or loss during the game. Each of these aspects would add another function or object to our programs, and had to be carefully considered.

Evaluating:

After all considerations were established and we decided to begin construction of our project, we needed to understand how to achieve each of these tasks. The first step in construction was the visual representation of the game without any dependent structures or function calls.

The grid was overall simple enough, but the complications arose when we decided to approach this with an object oriented perspective. This was a fairly new approach to Javascript programming and we needed to identify how exactly each part of this grid would be constructed within an object, which we named Minesweeper. Allowing all of our dependent variables to be held within a single Minesweeper object made debugging and overall oversight of the project become a very manageable task. The grid was a two-dimensional array within the Minesweeper object, which then consisted of Cell objects. Each of these objects were stitched together with embedded functions and variables that assigned properties to the html table as well as altered the respective variables such as the two dimensional array and the visible cell whenever a player clicked on a cell.

The other portion of our project focused on php server/client data manipulation and being able to store that information into a database. We knew we wanted to have a competitive nature added to our game, thus a username and related account were to be generated to allow scores to be recorded into a leaderboard. Once we had the server setup to receive the information from the user, and check all input with RegEx, we stored what we required into a MySQL database for later use.

Creating:

From this project we have a new understanding of how to implement object oriented programming into multiple languages even though they might be outside of our comfort zone. Modern Javascript and PHP have an architecture which supports object oriented programming and can make their otherwise foreign layouts, appear to be familiar and adaptable through similar coding practices.

Designing the project would have been done likely all through Javascript as it is entirely feasible to do so. However, with the values we have taken from this course we are able to construct an entire project which successfully and securely manages a client/server relation. PHP, Javascript, JQuery, Ajax, MySQL, and CSS were all used in unison to allow this project to flow together as one element.

From this project we are now able to see how multiple languages being combined are what allow for web development to succeed. As server side programming requires the understanding and interpretation of client side input, so does client side programming require the understanding of server side requirements for input and then knowing what can be done with the feedback after a call is made to the server.