Mine Sweeper Challenge - Ali Johnson

When approaching challenges like this, I list the main requirements, look at the problems that need to be solved, and then, if needed, break the problems down further. Here is how I did this for the mine sweeper challenge.

Creating the custom-sized game board

- A 2D array seemed like the obvious choice to store the data. Hard coding a 2D test array confirmed I could easily access and change data within it.
- I then made an array of empty strings for a row, using a for-loop to set the length of the array to equal the number of required columns.
- To create the 2D array for the game board, I used another for-loop to push multiple row arrays into another empty array, again, using a for-loop to set the number of rows required.

Adding the mines randomly to the empty game board.

- First, I tried to randomly add one mine to the game board. I did this by creating a function that generated a random index number. Then, using this, I could access a random field on the game board which I could assign an “X” to represent a mine.
- To add multiple mines, I knew I would require a loop, a condition to ensure no field had more than one mine added, and another condition to break the loop when enough mines had been added.
- For this, I wrote a conditional ternary operator to check for multiple mines, which I then used in a while-loop. As each mine was added, I increased the value of a counter which broke the loop when sufficient mines had been added.

Adding the clue numbers.

- To solve this requirement I realised I needed the following...
  - To loop through and find every mine, find all adjacent fields to that mine, find the values within those fields, add 1 to each field to update the clue value.
- To loop through and find the mines, I created a nested for-loop to loop through the 2D array using a conditional ternary operator to check if a mine was in each field.
- To update the clue value of a field, I wrote a function that increased the value of a field by 1 (or replaced it with 1 if an empty string).
- Another function found all surrounding fields for a given mine and updated the clue value with the above clue value function.
- Putting these functions together, I could successfully update the field clue values, except when a mine was on the border of the array. In this case the function broke as it looked for non-existent, negative index values.
- To solve these border error issues I wrote a conditional statement that returned true if the index values were valid, however, due to time constraints I didn’t quite get this working.