Project 2B: Javascript Game: WordGuess

CSE 100/INFO 100 Fluency with Information Technology

Project 2B

Part B is worth **70 points** and is due on Sunday, May 21, 2005, before 10 pm. It will consist of a file called **project2b.html**. Please turn this file into Catalyst, as well as place it under the directory **public_html/fit100/project2b** in your Dante home directory, as with Project 1.

1. Copy the work you did in project 2A to project2b.html, changing any references to "2A" from "2A" to "2B". Also, get rid of the `alert` boxes from Part A.
2. As with part A, make sure you validate your page, write new thoughts in your "thoughts" section, and document all your JavaScript code in the same manner as before.
3. Now write the `checkGuess()` function, which we started in Part A. The `checkGuess()` function is run only after the player has entered a guess. This is pseudo-code for the `checkGuess()` part of the project:

```
Create variable previousGuesses and set it equal to an empty string.
Create variable totalFound and set it equal to 0.

Begin checkGuess function:
      Create variable found and set it equal to 0.
      Read in the value of our input and store it to variable guessedLetter.
      Change guessedLetter to upper case.
      For each index i from 0 until the length of previousGuesses
       (i.e. for each character in previousGuesses),
            If guessedLetter same as character at index i of previousGuesses,
             (i.e. guessedLetter equals current character)
                  Show alert box telling user they've guessed this letter before.
                  Set found variable to -1.
      If found variable is not -1,
            Add guessedLetter to list of previousGuesses.
            For each index i from 0 until the length of secretWord
             (i.e. for each character in secretWord),
                  If guessedLetter equals character at index i of secretWord,
                        Write the guessedLetter to the corresponding blank line
                        Add one to the found variable.
            If found is 0 (i.e. if nothing was found),
                  Add guessedLetter to the HTML area for wrong guesses.
                  Call a function to color in another box of the progress bar.
            Else
                  Add found to totalFound.
                  If totalFound is equal to the length of secretWord,
                        Display an alert box that tells the player they've won.
                        Refresh the page for the user, so they can start over.
End checkGuess function.
```

The purpose of the `checkGuess()` function is to take the letter the player just entered, and check whether it has already been guessed. If it hasn't been guessed, we are going to check if it is actually in our secret word by looping through each character of our secret word and matching it with the guessed letter. Whenever it is found, we replace the appropriate blanks, and we also keep track of the number of times that letter was found. If it isn't found, then we write it to the incorrect guesses area of the webpage, and move the progress bar. But if it was found, we check to see if the player has guessed all the characters in the code. If so, they win.

a. First of all, delete the old alert box we had there, if you haven't already gotten rid of it.
b. We need to create a variable *outside* of our `checkGuess()` function that will store all the guessed letters. We'll call this variable `previousGuesses`, and for now, it will be empty, i.e. it will be an empty string, denoted as two single quote marks with nothing inside. Do this by adding the following line in your head near the top of your code, right after you declared your `secretWord` variable.

   ```
   var previousGuesses = '';
   ```

c. Now go back inside the body of your `checkGuess()` function. We will create a variable called `found`. It will keep track of the status of your search for the guessed letter. If we find that the player has already guessed a certain letter, we will store a –1 (negative one) inside the `found` variable. If this is the player's first guess at a letter and it has not been found in the secret word, the value of `found` will be 0 (zero). Otherwise, the value of `found` will be the number of times it has currently been found in the secret word. In other words, at the end of our search, if "little" was the secret word, and the player guessed "t" then the value of `found` is 2. Create your `found` variable, and assign 0 (zero) to it for now. We haven't started searching yet, so obviously, we've found the guessed letter zero times.
d. Within the `checkGuess()` function, create a variable called `guessedLetter`. Assign to this variable the user's input. You should know how to do this from Lab 6.
e. For simplicity, we are only going to deal with upper case letters in this game. If you look at the `pickWord()` function we created last time, we used a built-in function called `toUpperCase()` to convert our selected secret word to upper case. Use this function again to change your `guessedLetter` to upper case, and assign the result back to `guessedLetter`. This should be one line of code.
f. Since we have already stored the player's guess in our `guessedLetter` variable, we can now erase whatever the user entered in the box. Do this by adding the following line ***after*** the code

we just wrote above:

```
document.getElementById('guess').value = '';
```

g. Now, write a `for` loop that loops through the length of `previousGuesses`. We will use this loop to go through each letter stored in `previousGuesses`, i.e. look through all the previous guesses. The `previousGuesses` variable is not necessarily empty, even though we initialized it to be so at the beginning, because each time the `checkGuess()` function runs, a letter will be added to `previousGuesses` (which we'll do later), so there might be letters in there. If you've forgotten how to write a `for` loop, look back at the `writeBlanks()` function you wrote last time, and change the appropriate thing(s). Your loop variable should still be `i` though. (Loop variables are named `i` out of convention. Technically, you could name it anything and you'd be fine. But for this project, don't.)

h. Inside the body of the `for` loop you just created, write an `if` statement (no else block needed) that checks if the `guessedLetter` is equal to the character that is at the current index `i` of `previousGuesses`, in other words, we're checking if the user's guess has already been guessed. To do this you need to use the `charAt()` function. To figure out how to use the function, look at Homework 3. Next, inside the body of the `if` statement, include two statements.

   i. Show an alert box that tells the user that they guessed this letter before.

   ii. Change the `found` variable to be –1 (negative one), indicating that this letter they guessed has already been guessed. If you like, though this is optional, you may also *add* to the continue condition of the `for` loop you wrote that you should only continue while `found` is still 0, i.e. `found` is not –1. This way, we stop the `for` loop from looping again.

i. Test what you just wrote by changing your variable declaration of `previousGuesses` at the very top of your code to be some list of letters, e.g. `'ABCDE'` instead of the empty string `''` and then try entering guesses into the input box. When you enter a letter that is in that list, you should receive an alert box saying that you have already guessed that letter. Otherwise, nothing should happen. If this doesn't happen, debug your work (and don't forget to write down any observations in your "Thoughts" section). After you are satisfied with your program thus far, put the empty string back.

j. Now, *after* your `for` loop, write an `if` statement that executes if `found` is *not* –1, in other words, it will run as long as we've found that the player's current guess was not in the list of `previousGuesses`.

k. We will now write quite a few lines of code, all of which will be *within* the body of the `if` block you just created. This is because if the letter was already found to have been guessed, we're done, we don't need anything else to happen. So, everything inside the `checkGuess()` function from now on will be inside the `if` block you just created, and I'm not going to write explicitly where to put the following code, unless it is in some place different.

l. Add the current guess to the list of `previousGuesses`. This should be one line of code.

m. Write a `for` loop that iterates through each character of `secretWord`. Inside this `for` loop, write an `if` statement that checks if the `guessedLetter` is found at the current letter of `secretWord` whose index we're looping through. If it is, show an `alert` box that shows what letter we've just found. Test your code. If it doesn't work, debug, and keep notes about it.

n. Once you are done debugging, remove the `alert` box that you created in the previous instruction.

o. In place of where that `alert` box used to be, we are going to replace the `innerHTML` of one of the blanks we created earlier (from project2a) with the `guessedLetter` (since it's correct). This can be done with a line of code that will soon follow. After you add it, test your code to be sure that the blanks are overwritten when you guess a word correctly. This may require that you also show an `alert` box somewhere showing what your secret word is, or else you won't really know if you're writing the correct word. Since you're being given this line of code for free, write a comment in your code above it that explains *in detail* what every part of this statement does.

```
document.getElementById('blank' + i).innerHTML = guessedLetter;
```

p. Directly beneath the line of code you just added, inside the `if`-statement that checks whether the guessed letter is correct, *increment* the `found` variable. This means to add one to it, and store it back to itself. We're doing this to keep track of how many times we've found the guessed letter. So if this was the first time we've found the `guessedLetter` in our `secretWord`, the `found` variable will change from `0` to `1`, whereas if this is the twelfth time you've found it (assuming it appears that often in your `secretWord`), you will probably be increasing the value of your `found` variable from `11` to `12`. Test this a few times by showing an alert box with the value of your `found` variable each time you change it. Remove the box when you are done.

q. Using our `for` loop, we've now gone through each letter of our secret word and checked whether the guessed letter matches any of them. Now, outside of the `for` loop, check to see if any letters were found by writing an `if-else` statement.

i. The `if`-block should execute if nothing was found; use the `found` variable to determine this. Inside the `if`-block, we're going to add the `guessedLetter` to the list of incorrect guesses on our HTML document.

ii. Create a paragraph with an `id` attribute of "`wrongGuesses`" in the body of your document, wherever you want your list of wrong guesses to show up. Leave the paragraph element empty for now, but in front of the paragraph, create another paragraph that says "Wrong guesses:"

iii. Now, back to that if-block we were writing. Add the `guessedLetter` to the `innerHTML` of the paragraph element with the `wrongGuesses` `id`. Remember to *add* to it, not *replace* what was there. Test to see if your program works; debug.

iv. If the player guessed wrong, we also want to move the progress bar. Inside this `if`-block, call the function `moveProgressBar()`. This function hasn't been written yet, we will write this function later. For now, let's comment out this line, but keep it there as a placeholder so we'll remember where in the code we need to move the progress bar.

v. The `if`-block executes when the letter is not found, which means the `else`-block of this `if-else` statement executes when the `guessedLetter` *is* found. In the `else` block, we want to check if all the letters have now been found, because if so, that means the player has won. To do this, we are going to keep track of the number of letters that have been found in the entire game (not just for this `guessedLetter`) so far, and see if it is the same as the number of letters in the secret word. If so, that means that all the letters in our secret word have been found.

   1. Create a variable *outside of the `checkGuess()` function* at the top of your code (beneath your `secretWord` and `previousGuesses` variable declarations), called `totalFound` that is initialized to 0 (zero).
   2. Now, back inside our `else`-block, increase the value of `totalFound` by adding the value of `found` to `totalFound`.
   3. Next, still inside the `else`-block, write another `if`-statement that checks if `totalFound` is equal to the length of your `secretWord`. If so, display a little alert box telling the user that they've won. Also refresh the page for them using the following bit of code:

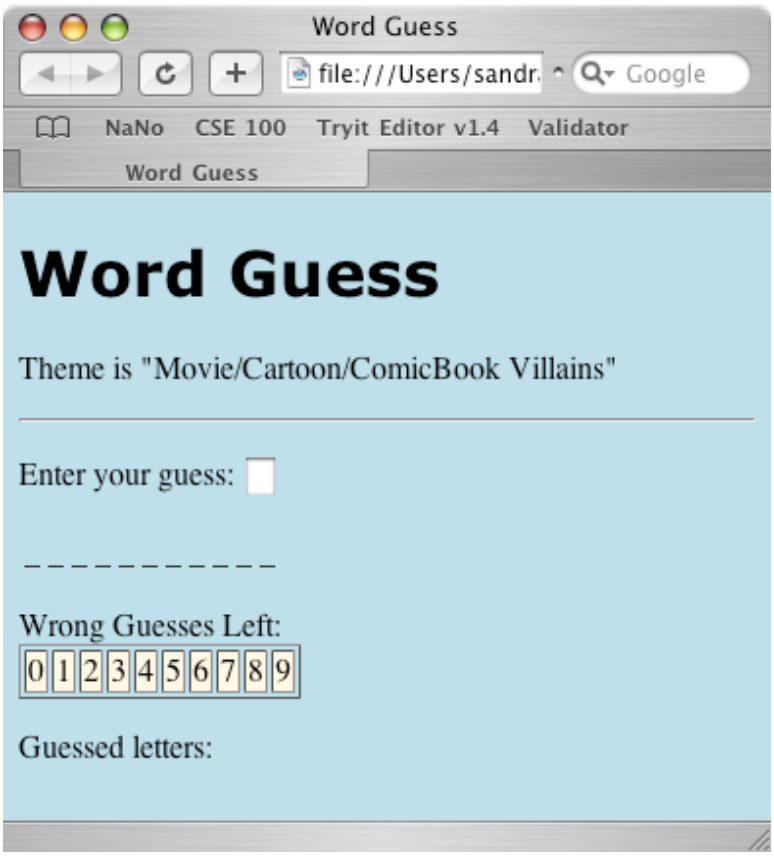
      ```
      window.location.reload();
      ```

r. Test and debug your program. It should now do everything except for the progress bar part. The game isn't going to count how many wrong guesses you have. In other words, you can't lose the

game, you can only win. ☺ Play with your game for a while to make sure it works. You're done with the `checkGuess()` function! Celebrate, take a break, and then come back and work on the exciting next part.

4. We are now going to work on the progress bar. We will create two functions to do this. First, we'll create a function that creates our progress bar. This function will be called `createProgressBar()`. (I know, very creative. In programming, the less creative your names, the better.) Then we will create a function that moves the progress bar, i.e. colors in the boxes of the progress bar when the player guesses incorrectly. When the progress bar reaches the end, this function will display an alert box telling the player that they've lost, and then it will refresh the page. This function will be called `moveProgressBar()`. You should recognize this function, it was the function that we called and commented out earlier.

    a. The `createProgressBar()` function is very similar to the `writeBlanks()` function from project 2A. Add a call to it in the `<body>` of your document, and then write the function yourself. The function should use a `for` loop to create ten cells of a table. Do not use the literal value of 10 in your for loop, create a variable outside your function (up where all the other variables were declared) called `size` and set it equal to the number of boxes we want our progress bar to have, and then within your function, loop using that variable. You can get the borders to show up by adding a `border` attribute to your `<table>` tag and setting the attribute to 1 (one). Also, this time, instead of assigning the `id` attribute of the cells to `blankx`, assign them to `progressx`, where x stands for the box's number, starting from 0 and going to 9. Here's a picture to refresh your memory of what the progress bar should look like. The progress bar is directly under the words "Wrong Guesses Left:". You may add those words there too, if you like. `Test your createProgressBar()` function, and debug. Don't forget to add notes to your "Thoughts" section, if necessary. You should be doing this whenever you debug.



b. Finally, let's write our last function, the `moveProgressBar()` function.
   i. Uncomment the line where you commented out the `moveProgressBar` function earlier.
   ii. Create a variable called `progressLocation`, initialized to 0 (zero). This variable should be in the same place as all your other variables near the beginning of the head of your document.
   iii. Create a new function in the head of your HTML document called `moveProgressBar()`.
   iv. In the body of your function, create a variable called `currentBox` and assign to it the table data (cell) element that has an id that contains the value of `progressLocation`. In other words, we are setting our `currentBox` to be the next box we have to color. For a hint on how to access the element with the correct `id` attribute, check out the line of code I gave you in part o (o as in the letter of the alphabet, o) above, for `checkGuesses()`. You should not be using the exact same line, because you are not doing anything with `innerHTML` here, but rather the entire element itself, but it should give you a clue on how to write this one.
   v. We want to take this box and color it. Do so with the following line. You may replace black with any color you'd like.

```
currentBox.style.backgroundColor = 'black';
```

   vi. Increment `progressLocation` by 1 (one).

vii. If your `progressLocation` is now equal to the size of your progress bar, show an alert that tells the player they've lost and what the correct word was, and refresh the page for them.

5. Test your program, debug, write down your debugging process, and document your code.
6. You may make some style changes to your document if you like, using the HTML skills you learned from project 1. You're done!

Grading

For project 2B, you will be graded as follows:

1. 5 points for correct turn-in and validation.
2. 5 points for commenting your code
3. 5 points for your "Thoughts" section
4. 25 points for your `checkGuess()` function, and related variables and HTML elements.
5. 30 points for your `createProgressBar()`, `moveProgressBar()` functions, and related variables/HTML elements.

…for a total of 70 points. You may get extra credit as well. Details in the next section.

Extra Credit (0-5 points, due date same as Project 2B)

You're done with your lab, but it's pretty boring looking, isn't it? Why not spice it up? Here are two suggestions for extra credit points. Or think of something on your own, but check with your TA to be sure it's a reasonable idea first. Don't start on the extra credit until you've finished your project, though.

You can get at most 5 points extra credit all together. You may get a 0 if you attempt extra credit, but don't get it right. You will probably get 2 or 3 points if you do a decent job. You will get 5 points only if you do a spectacular, incredible, jaw-dropping job, or if you do more than one piece of extra credit, and do them decently. But think how cool your program will be!

1. Instead of coloring table rows when the player guesses incorrectly, display a series of pictures instead. For instance, you can have a series of pictures of an outdoor scene at different times during the day, where the sun moves across the sky. When your pictures reach nighttime, the player loses. You could display your pictures in the progress bar, or you could have an area where you show just a picture. Lab 7 should be helpful in figuring out how to do this. Include your picture files in your Catalyst turn-in.
2. Currently, our secret words can only be words—we can't use phrases because we can't handle spaces in our "secret word." Change your program such that your secret words can have spaces. The user should not have to guess the spaces, they should be already displayed for them, instead of the blank lines. Your program should still work correctly in all cases.