Assignment 2: The Simon game

Objective: Students will practice

- Problem-solving
- JavaScript functions
- JavaScript DOM and Event handler

Assignment instruction

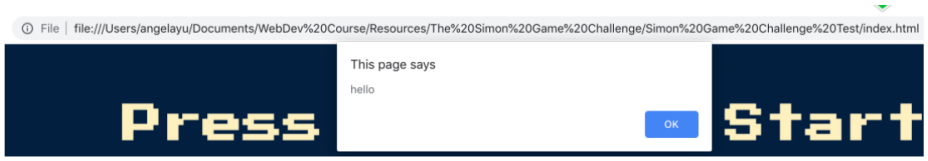
- The instruction, clip of 'how to play the Simon game', and file resources are posted on MS Teams channel **Assignment**– the Simon game of subject 953262 (your section).
- Download the zipped file of resources.
- There are **10 steps** according to the instruction sheet posted on the channel.
- The assignment is worth 23 points in total **which is calculated as 5% of your final grade.**
- Score criteria: full point (for output correct); -1 (for output does not correct); -1 (for not follow problem constraint
- Assignment Submission
    - Upload your solutions to MS Team assignments. **The submission 'due date' and 'close date' is on midnight Tuesday, 13th February 2024.**

Step 0: Download the starting files.

In the project folder, you will find **sounds folder** containing the UI sounds for all four buttons, **index.html** file that has four buttons pre-styled with **CSS**, and a **video clip** showing how the Simon game is played. **Watch the clip before you start implementing the game.**

Step 1: Add JavaScript and jQuery (1 point)

1. Create a new file called **game.js**
2. Link to this new external JS file from your index.html
3. Add an **alert** to game.js and test that the alert gets triggered when you load up index.html in Chrome. Once, you've confirmed that game.js is correctly linked, you can delete or comment out the alert.



Step 2: Create a new pattern for a user to memorize (6 points)

1. Create global variables to be used in the game as follows.
    1. A new **array** called `buttonColours` and set to hold the sequence "red", "blue", "green", and "yellow".
    2. A new **empty array** called `gamePattern`.
    3. A new **empty array** called `userClickedPattern`.
    4. A new game started **variable** called `started` and set to false.
    5. A new level **variable** called `level` and set to 0.
2. Use a JavaScript event handler to detect when a keyboard key is pressed, triggering the execution of the `` `nextSequence()` `` function, the details of which are explained below.
3. The `nextSequence` function is designed to set up the next level in the game, providing the user with a new pattern to memorize, updating the level display,

and adding the correct color to the game pattern. Additionally, it includes visual and auditory cues to enhance the user experience. Here are the steps for implementing code inside the `nextSequence` function.

1. Clears the `userClickedPattern` array to keep track of the user's clicked patterns for each game level.
2. Increments the `level` variable by 1, indicating that the game is progressing to the next level.
3. Updates the text content of an HTML element with the ID "`level-title`" to display the current level.
4. Generates a random number between 0 and 3 (inclusive). This is used to randomly select a color from an array of the `buttonColours`.
5. Create a new **variable** called `randomChosenColour` to store a color from the `buttonColours` array based on the randomly generated number from the previous step (4). This color is the one that the user needs to memorize and replicate.
6. Adds the randomly chosen color to an array called `gamePattern`, which likely represents the correct pattern that the user needs to match.
7. Log the value of the `gamePattern` array showing the result of gamePattern.
8. Invoke a given function `fadeInOut` to perform a brief fade-out and fade-in animation on the HTML element corresponding to the randomly chosen color. This visual effect may serve as a visual cue for the user. *(hint: use `getElementById()` to access to the randomly chosen color HTML element)*

```
function fadeInOut(element, duration) {
  // Fade in
  element.style.opacity = 0;
  var fadeInInterval = setInterval(function () {
      element.style.opacity += 1;
```

```
            if (element.style.opacity >= 1) {
                clearInterval(fadeInInterval);

                // Fade out
                var fadeOutInterval = setInterval(function () {
                    element.style.opacity -= 1;
                    if (element.style.opacity <= 0) {
                        clearInterval(fadeOutInterval);

                        // Reset opacity for next fadeIn
                        element.style.opacity = 1;
                    }
                }, duration / 10);
            }
        }, duration / 10);
    }
```

9. Use Google/Stackoverflow to figure out how you can use JavaScript to **play the sound** matching to the `randomChosenColour`.
10. Run the Simon game on the live server and press any key to see a log of `gamePattern` array in the Chrome Developer Tools console and listen to the matched `randomChosenColour` sound.

Step 3: Check which button is clicked. (2 points)

1. Use a JavaScript event handler to detect when any of the buttons are clicked and trigger a handler function. *(Hint 1. Use method querySelectorAll() and forEach())*
2. Inside the handler, create a new **variable** called `userChosenColour` to store the id of the button that got pressed. *(Hint 2. Use target property of event object.)*

--> So if the Green button was clicked, `userChosenColour` will equal its id which is "green".

```
<div type="button" id="green" class="btn green">
```

3. Add the `userChosenColour` to the **array** called `userClickedPattern`, which likely represents the pattern that the user has chosen the colors so far. At this stage, if you log the `userClickedPattern` you should be able to build up an array in the console by clicking on different buttons.

Step 4: Add Sounds to the user's Button clicks (2 points)

4. In the same way, we played sound in `nextSequence()`, when a user **clicks on a button**, the corresponding sound should be played. e.g., if the Green button is clicked, then green.mp3 should be played.

1. Create a new function called `playSound()` that takes a single **input parameter** of the sound name.
2. Take the code we used to play sound in the `nextSequence()` function and move it to `playSound()`.
3. **Refactor** the code in `playSound()` so that it will work for both playing sound in `nextSequence()` and when the user **clicks a button.**

Step 5: Add animations to the user's Button clicks (2 points)

1. Create a new function called `animatePress()`, it should take a single input parameter called `currentColour`.
2. Look inside the styles.css file, you can see there is a **class** called `pressed`, it will add a box shadow and change the background color to grey.
3. Use **JavaScript to add this pressed class** to the button that gets clicked inside `animatePress()`.
4. Use Google/Stackoverflow to figure out how you can **use JavaScript to remove the pressed class after 100 milliseconds.** *(Hint. https://www.google.com/search?q=how+to+add+delay+javascript)*

Step 6: Start the game (1 point)

1. **Add game logic to keep track of whether the game has started or not** into the handler when the user presses any key to start the game (at Step 2.2). While matching the pattern, the user cannot press any key until the game is over. *(Hint. Use* **variable** `started`)

Step 7: Check the User's Answer Against the Game Sequence (4 points)

1. Create a new **function** called `checkAnswer()`, it should take one **input** with the name `currentLevel`. The `checkAnswer()` will be invoked after a user has clicked any color button, passing in the index of the `currentLevel` into in the `function`. *E.g. If the user has pressed red, green, red, yellow, the index of the last answer is 3.*
2. Then write **an if statement** inside `checkAnswer()` to check if the currentLevel of `userClickedPattern` array is the same as the `gamePattern` array. If so then log "**success**", otherwise log "**wrong**".
3. You can now use these log statements along with logging the values of `userClickedPattern` and `gamePattern` in the Chrome Developer Tools console to check whether your code is performing as you would expect and debug your code as needed.
4. If the user correctly answered the most recent step in step 3, further verify their sequence completion with **an additional if statement**, comparing the lengths of `userClickedPattern` and `gamePattern` to ensure they match. If so then invoke `nextSequence()` after a 1000 millisecond delay.

Step 8: Game Over (2 points)

1. In the sounds folder, there is a sound called **wrong.mp3**, play this sound if the user gets one of the answers wrong. (hint. continue working on **an else statement** inside `checkAnswer()`)

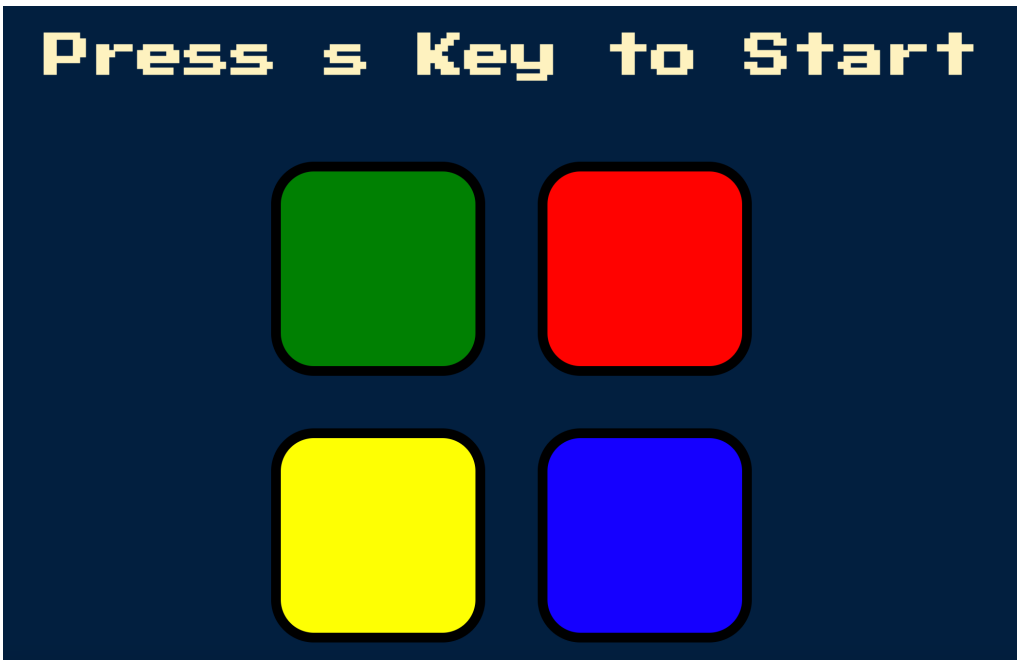
2. In the styles.css file, there is a class called "**game-over**", apply this class to the **body** of the website when the user gets one of the answers wrong and then **removes it after 300 milliseconds.**
3. Change the **text content of the HTML element title** to say "**Game Over, Press Any Key to Restart**"

Step 9: Restart the Game (1 point)

1. Create a new function called `startOver().`
2. Invoke `startOver()` if the user gets the sequence wrong of game pattern. Inside this function, you'll need to **reset** the values of `level`, `gamePattern` and `started`

Step 10: Modify how to start and restart the game (2 points)

On the game homepage, the HTML element is displayed '**Press s Key to Start**' and **to start the game, a user must press only 's' on the keyboard**.



When the game is over, the HTML element is displayed '**Game Over, Press r Key to Restart**' and **to restart the game, the user must press only 'r' on the keyboard, and the user resumes the game homepage**.