

Fundamentals of WEB Technologies

**Title: Sample Game: «Bulls & Cows»**

**Performed by:** Zhangazy Nurshapagat

**Checked by:** Shoiynbek Temirlan

Алматы, 2024

**Project objectives:**

The goal of this project is to create the "Bulls and Cows" game. The primary objective is to encourage users to develop their logical reasoning skills by guessing numbers. The game serves as both entertainment and a tool for enhancing analytical thinking.

**Project Features:**

1. **Game Rules:** The user must correctly guess a 4-digit number. Each digit can only appear once.
2. **Interactive Interface:** Users can input their guesses and immediately see the results (bulls and cows).
3. **Guess History:** Each guess's results are displayed on the screen, allowing users to track their previous attempts.
4. **Win Notification:** A congratulatory message is shown when the user guesses the number correctly.
5. **Restart Option:** A dedicated button allows the user to restart the game.
6. **Animations and Sounds:** Visual and audio elements are added to enhance user engagement.

**JavaScript Usage:**

1. **Game Logic Implementation:** The core game logic is developed using JavaScript, including:
   * Randomly generating the hidden 4-digit number.
   * Validating user guesses and calculating the results for bulls (correct digit and position) and cows (correct digit, wrong position).
2. **Handling User Interactions:**
   * Receiving and processing user inputs.
   * Displaying error messages for invalid inputs.
3. **Manipulating HTML Elements:**
   * Adding guess results to the history table.
   * Displaying a win message and enabling the restart functionality.
4. **Animations and Sounds:**
   * Managing the movement of visual elements and playing sounds to create engaging effects.

This document provides a concise overview of the project structure, its features, and how JavaScript is utilized to implement them effectively.

**User Guide: Examples of Use Cases**

**Objective of the Game:**

In the "Bulls and Cows" game, the user must correctly guess a 4-digit secret number. Each digit is unique and does not repeat.

**User Guide:**

1. **Starting the Game:**
   * The user inputs a 4-digit number in the game interface.
   * The number must consist of unique digits and exactly four digits.
2. **Feedback:**
   * After each input, the user receives feedback indicating the number of "bulls" and "cows":
     + **Bulls**: Correct digit in the correct position.
     + **Cows**: Correct digit but in the wrong position.
   * Example: If the secret number is "1234" and the entered number is "1325":
     + Bulls: 1 (The "1" in the first position).
     + Cows: 2 ("2" and "3", but their positions are incorrect).
3. **History:**
   * Each guess is added to a history table that displays:
     + The attempt number.
     + The entered number.
     + The number of bulls and cows.
4. **Winning:**
   * If the user guesses all 4 digits correctly, a victory message is displayed on the screen.
   * Victory message: "Congratulations! You guessed the number [secret number] in [number of attempts] attempts!"
5. **Restarting the Game:**
   * Using the "Restart" button, the game resets. All history and feedback are cleared.

**Examples of Use Cases:**

**Case 1: Valid Input**

* The user enters a number: "5678".
* Feedback: Bulls - 2, Cows - 1.
* The guess is recorded in the history table.

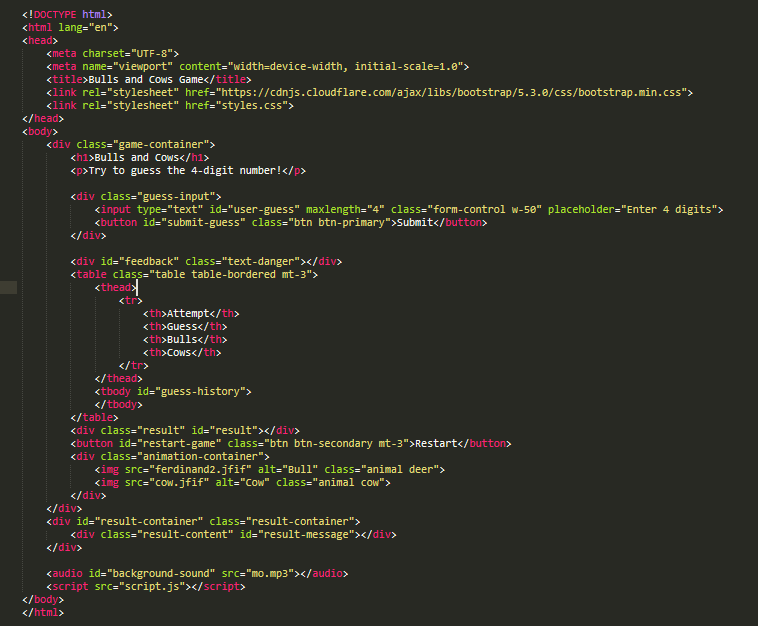
**Case 2: Invalid Input**

* The user enters an invalid number: "1123" (repeated digits).
* Feedback: "Please enter a valid 4-digit number with unique digits."

**Case 3: Winning the Game**

* The user correctly guesses the number: "1234".
* Message: "Congratulations! You guessed the secret number in 7 attempts!"
* The "Restart" option becomes available.

This guide provides users with all the necessary information to play the game effectively.



 **<!DOCTYPE html>**  
This line declares that the document uses the HTML5 standard. It informs the browser about the version of HTML being used.

 **<html lang="en">**  
This begins the main HTML element. The **lang="en"** attribute specifies that the document's primary language is English.

 **<meta charset="UTF-8">**  
Specifies the character encoding for the document as UTF-8, ensuring proper rendering of various characters.

 **<meta name="viewport" content="width=device-width, initial-scale=1.0">**  
Ensures the page scales appropriately on different devices.

* **width=device-width**: Sets the viewport to match the device's width.
* **initial-scale=1.0**: Sets the initial zoom level to 100%.

 **<title>Bulls and Cows Game</title>**  
Defines the title of the document, which appears on the browser tab.

 **<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap/5.3.0/css/bootstrap.min.css">**  
Links the Bootstrap CSS library for styling.

 **<link rel="stylesheet" href="styles.css">**  
Links to an external CSS file for custom styling.

 **<div class="game-container">**  
A container for the main game interface. It organizes the content and centers it on the page.

 **<h1>Bulls and Cows</h1>**  
Displays the game’s title in a large heading.

 **<p>Try to guess the 4-digit number!</p>**  
Provides a brief instruction for the game.

 **<div class="guess-input">**  
A block that holds the input field and the submit button.

 **<input type="text" id="user-guess" maxlength="4" class="form-control w-50" placeholder="Enter 4 digits">**

* **type="text"**: Allows text input.
* **id="user-guess"**: An identifier used to reference this element in JavaScript.
* **maxlength="4"**: Restricts input to 4 characters.
* **class="form-control w-50"**: Applies Bootstrap styling to the input field.
* **placeholder="Enter 4 digits"**: Displays placeholder text in the input field.

 **<button id="submit-guess" class="btn btn-primary">Submit</button>**  
The button used to submit the user’s guess.

* **id="submit-guess"**: A unique identifier for the button.
* **class="btn btn-primary"**: Adds Bootstrap styles for a primary button.

 **<div id="feedback" class="text-danger"></div>**  
Displays error messages for invalid input.

* **class="text-danger"**: Styles the text in red.

 **<table class="table table-bordered mt-3">**  
A table to display previous guesses and their results.

* **class="table table-bordered mt-3"**: Uses Bootstrap styles for a bordered table with margin at the top.

 **<thead>** and **<tr>**  
The table headers define the column names:

* **Attempt**: Number of guesses.
* **Guess**: The user's entered number.
* **Bulls** and **Cows**: Result columns indicating correct digits.

 **<tbody id="guess-history">**  
A placeholder for dynamically adding rows with guess history using JavaScript.

 **<div class="result" id="result"></div>**  
Displays the game’s result, such as winning or losing.

 **<button id="restart-game" class="btn btn-secondary mt-3">Restart</button>**  
A button to restart the game.

* **class="btn btn-secondary"**: Styled as a secondary button.

 **<div class="animation-container">**  
A container for the animated images.

 **<img src="ferdinand2.jfif" alt="Bull" class="animal deer">**  
Displays an image of a bull.

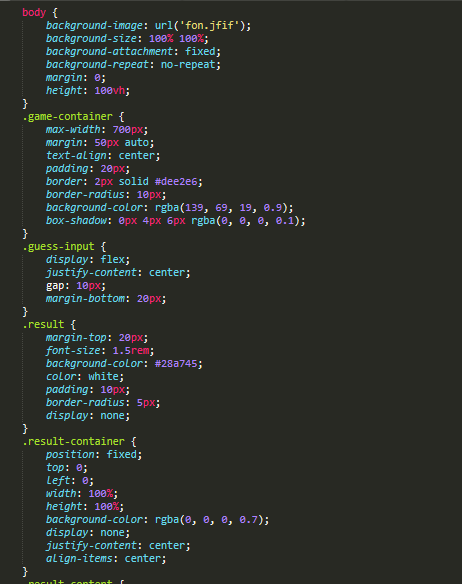
* **class="animal deer"**: Applies styles for animation.

 **<img src="cow.jfif" alt="Cow" class="animal cow">**  
Displays an image of a cow with similar animation.

 **<audio id="background-sound" src="mo.mp3"></audio>**  
Adds an audio file to play during the game.

 **<script src="script.js"></script>**  
Links the external JavaScript file containing the game logic.

**</body>** and **</html>**  
Marks the end of the document structure.



 **body {**

This starts the style for the body element.

 background-image: url('fon.jfif');

Sets the background image of the page to fon.jfif.

 background-size: 100% 100%;

Makes the background image cover the full width and height of the screen, stretching it to fit.

 background-attachment: fixed;

Fixes the background image, so it does not move when the user scrolls.

 background-repeat: no-repeat;

Prevents the background image from repeating. It will be displayed only once.

 margin: 0;

Removes any margin around the body, ensuring no space around the edges.

 height: 100vh;

Sets the height of the body to 100% of the viewport height (the full height of the screen).

** game-container {**

Starts the style for the main game container.

 max-width: 700px;

Sets the maximum width of the game container to 700 pixels.

 margin: 50px auto;

Centers the container on the page horizontally and adds 50 pixels of margin on the top and bottom.

 text-align: center;

Centers the text inside the container.

 padding: 20px;

Adds 20 pixels of padding inside the container.

 border: 2px solid #dee2e6;

Adds a 2-pixel solid border around the container with a light grey color.

 border-radius: 10px;

Rounds the corners of the container by 10 pixels.

 background-color: rgba(139, 69, 19, 0.9);

Sets the background color of the container to a semi-transparent brown.

 box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.1);

Adds a subtle shadow around the container to create depth.

** guess-input {**

Starts the style for the input area where users will enter their guesses.

 display: flex;

Uses a flexbox layout to align elements horizontally.

 justify-content: center;

Centers the items horizontally within the container.

 gap: 10px;

Sets a 10-pixel gap between the elements inside the container.

 margin-bottom: 20px;

Adds 20 pixels of margin below the input area.

** result {**

Starts the style for the result area where the game outcome is displayed.

 margin-top: 20px;

Adds a 20-pixel margin at the top of the result area.

 font-size: 1.5rem;

Sets the font size to 1.5 times the base font size.

 background-color: #28a745;

Sets the background color of the result area to green.

 color: white;

Sets the text color to white.

 padding: 10px;

Adds 10 pixels of padding inside the result area.

 border-radius: 5px;

Rounds the corners of the result area by 5 pixels.

 display: none;

Hides the result area by default.

** result-container {**

Starts the style for the full-screen result container.

 position: fixed;

Fixes the position of the result container so it stays in place on the screen.

 top: 0;

Positions the result container at the top of the screen.

 left: 0;

Positions the result container at the left of the screen.

 width: 100%;

Makes the result container take up the full width of the screen.

 height: 100%;

Makes the result container take up the full height of the screen.

 background-color: rgba(0, 0, 0, 0.7);

Adds a semi-transparent black background to the result container.

 display: none;

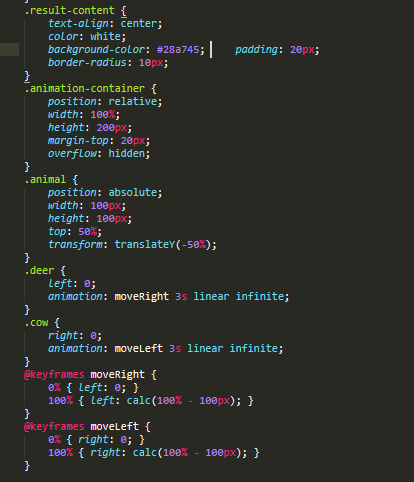
Hides the result container by default.

 justify-content: center;

Centers the content horizontally within the result container.

 align-items: center;

Centers the content vertically within the result container.



** .result-content {**

This starts the style for elements with the result-content class.

 text-align: center;

Aligns the text to the center inside the container.

 color: white;

Sets the text color to white.

 background-color: #28a745;

Sets the background color of the container to green (#28a745).

padding: 20px;

Adds 20 pixels of padding inside the container, creating space between the content and the edges.

 border-radius: 10px;

Rounds the corners of the container with a radius of 10 pixels.

** .animation-container {**

This starts the style for the animation container.

 position: relative;

Positions the container relative to its normal position, allowing child elements to be positioned absolutely.

 width: 100%;

Sets the width of the container to 100% of the available space.

 height: 200px;

Sets the height of the container to 200 pixels.

 margin-top: 20px;

Adds 20 pixels of margin at the top of the container.

 overflow: hidden;

Ensures that any content that overflows the container's boundaries is hidden from view.

** .animal {**

This starts the style for the animal images (such as the deer and cow).

 position: absolute;

Positions the element absolutely within its closest positioned ancestor (or the document body if no ancestor is positioned).

 width: 100px;

Sets the width of the element (the animal image) to 100 pixels.

 height: 100px;

Sets the height of the element to 100 pixels.

 top: 50%;

Positions the element vertically at 50% from the top of the container.

 transform: translateY(-50%);

Translates the element 50% upwards from its original position, effectively centering it vertically.

** .deer {**

This starts the style for the deer image.

 left: 0;

Positions the deer image at the left edge of the container.

 animation: moveRight 3s linear infinite;

Applies an animation named moveRight, which takes 3 seconds per cycle and repeats infinitely, moving the element from left to right.

** .cow {**

This starts the style for the cow image.

 right: 0;

Positions the cow image at the right edge of the container.

 animation: moveLeft 3s linear infinite;

Applies an animation named moveLeft, which takes 3 seconds per cycle and repeats infinitely, moving the element from right to left.

** @keyframes moveRight {**

Defines the keyframes for the moveRight animation.

 0% { left: 0; }

At the start of the animation, the element is at the left edge (0 pixels from the left).

 100% { left: calc(100% - 100px); }

At the end of the animation, the element is at the right edge of the container, with a 100px gap from the edge.

** @keyframes moveLeft {**

Defines the keyframes for the moveLeft animation.

 0% { right: 0; }

At the start of the animation, the element is at the right edge (0 pixels from the right).

 100% { right: calc(100% - 100px); }

At the end of the animation, the element is at the left edge of the container, with a 100px gap from the edge.



 **let secretNumber = "";** – Variable to store the secret number.

 **let attempts = 0;** – Variable to count the number of attempts.

** function generateSecretNumber() {** – Function to generate a secret number.

const digits = []; – Empty array to store the digits.

while (digits.length < 4) { – Loop to collect 4 digits.

const randomDigit = Math.floor(Math.random() \* 10); – Generate a random digit.

if (!digits.includes(randomDigit)) { – Check if the digit is not already in the array.

digits.push(randomDigit); – Add the digit to the array.

} – End of the loop.

return digits.join(""); – Convert the array to a string and return it.

** function initializeGame() {** – Function to initialize the game.

secretNumber = generateSecretNumber(); – Generate the secret number.

attempts = 0; – Set the attempt count to 0.

document.getElementById("guess-history").innerHTML = ""; – Clear the guess history.

document.getElementById("feedback").textContent = ""; – Clear the feedback message.

document.getElementById("result").textContent = ""; – Clear the result message.

document.getElementById("user-guess").value = ""; – Clear the user’s guess input.

document.getElementById("submit-guess").disabled = false; – Enable the submit button again.

document.getElementById("result-container").style.display = "none"; – Hide the result screen.

** function evaluateGuess(guess) {** – Function to evaluate the guess.

let bulls = 0; – Counter for the correct digits in the correct position (bulls).

let cows = 0; – Counter for the correct digits in the wrong position (cows).

for (let i = 0; i < guess.length; i++) { – Loop to compare each digit of the guess with the secret number.

if (guess[i] === secretNumber[i]) { – If the digit matches the secret number at the same position.

bulls++; – Increment the bulls counter.

} else if (secretNumber.includes(guess[i])) { – If the digit is in the secret number but at a different position.

cows++; – Increment the cows counter.

return { bulls, cows }; – Return the counts of bulls and cows.



** document.getElementById("submit-guess").addEventListener("click", () => {** – Adds an event listener to the "submit-guess" button, which runs the following function when clicked.

**const userGuess = document.getElementById("user-guess").value;** – Gets the user's input (the guess) from the input field.

**const feedback = document.getElementById("feedback");** – Selects the feedback element where error messages or feedback will be displayed.

** if (userGuess.length !== 4 || new Set(userGuess).size !== 4 || isNaN(userGuess)) {** – Checks if the user's input is invalid:

The length of the guess is not 4 characters.

The guess contains duplicate digits (using Set to check for uniqueness).

The guess is not a number.

** feedback.textContent = "Please enter a valid 4-digit number with unique digits.";** – If the input is invalid, display this message in the feedback area.

** return;** – Stop the function and prevent further actions if the guess is invalid.

** feedback.textContent = "";** – Clear the feedback area if the input is valid.

** attempts++;** – Increment the attempt counter.

** const { bulls, cows } = evaluateGuess(userGuess);** – Call the evaluateGuess function to calculate the bulls (correct digits in the correct position) and cows (correct digits in the wrong position).

** const historyRow = document.createElement("tr");** – Create a new table row (<tr>) for the current guess history.

** historyRow.innerHTML = \<td>${attempts}</td><td>${userGuess}</td><td>${bulls}</td> <td>${cows}</td>`;`** – Add the attempt number, guess, bulls, and cows to the row.

** document.getElementById("guess-history").appendChild(historyRow);** – Add the new row to the history table (guess-history).

** if (bulls === 4) {** – If the user has correctly guessed all 4 digits (4 bulls).

**document.getElementById("result").textContent = \Congratulations! You guessed the number ${secretNumber} in ${attempts} attempts.`;`** – Display a congratulatory message with the secret number and number of attempts.

**document.getElementById("submit-guess").disabled = true;** – Disable the "submit" button to prevent further guesses.

**document.getElementById("result-container").style.display = "flex";** – Show the result screen with the message.

**document.getElementById("result-message").textContent = \Congratulations! You guessed the number ${secretNumber} in ${attempts} attempts.`;`** – Display the same congratulatory message in the result container.

** const backgroundSound = document.getElementById("background-sound");** – Select the audio element for the background sound.

** backgroundSound.currentTime = 0;** – Reset the sound to the beginning.

** backgroundSound.play();** – Play the background sound when a guess is submitted.

** document.getElementById("restart-game").addEventListener("click", () => {** – Adds an event listener to the "restart-game" button, which triggers the following function when clicked.

**initializeGame();** – Calls the initializeGame() function to reset the game to its initial state (generates a new secret number, resets the attempts, and clears the guess history).

** document.getElementById("result-container").addEventListener("click", () => {** – Adds an event listener to the result container, which triggers the following function when clicked.

**document.getElementById("result-container").style.display = "none";** – Hides the result container when clicked, effectively closing the result pop-up.

** initializeGame();** – Calls the initializeGame() function when the script first loads to set up the game.