Critical Hit and Fumble Roller Overview

Created by **Brian Immel**, last modified on Aug 27, 2025

This guide describes the user interface and core functionality of the **Critical Hit and Fumble Roller** application. It breaks down the application's interactive elements and their use, so you can understand and operate its features effectively.

Application Overview

The Critical Hit and Fumble Roller provides specific outcomes based on user input or random generation. The application divides its functionality into two independent sections: **Critical Hit** and **Fumble**.

Critical Hit and Fumble Sections

Both sections share the same two core features: a **Random** option to generate an outcome and a **Submit** option to finalize a result. The **Critical Hit** section is for high-impact results, while the **Fumble** section is for low-impact or negative outcomes.

Random Roll Generation

The **Random** button generates a roll without needing any user input. Simply click it to get a result.

Submitting a Roll

The **Submit** button processes the currently displayed or selected outcome. Click it to finalize the result.

JavaScript Explained

This section details the JavaScript code that handles roll calculations and displays results. It covers selecting HTML elements, using event listeners for user interaction, and implementing logic to process roll values.

HTML Element Selection

This code gets programmatic access to specific HTML elements using their unique IDs.

```
const criticalHitBtn = document.getElementById('criticalHitBtn');
const fumbleBtn = document.getElementById('fumbleBtn');
```

```
const criticalHitDiv = document.getElementById('criticalHit');
const fumbleDiv = document.getElementById('fumble');
const criticalHitRandomBtn = document.getElementById('criticalHitRandom');
const criticalHitInput = document.getElementById('criticalHitInput');
const criticalHitSubmitBtn = document.getElementById('criticalHitSubmit');
const criticalHitResultDiv = document.getElementById('criticalHitResult');
const fumbleRandomBtn = document.getElementById('fumbleRandom');
const fumbleInput = document.getElementById('fumbleInput');
const fumbleSubmitBtn = document.getElementById('fumbleSubmit');
const fumbleResultDiv = document.getElementById('fumbleResult');
```

Explanation

This code initializes constants that store references to HTML elements by their ID. This is essential for JavaScript to interact with the page's buttons, inputs, and display areas.

Event Listeners for Section Toggling

These listeners switch the visibility of the Critical Hit and Fumble sections. Clicking a button makes its corresponding section visible while hiding the other.

Critical Hit Button Listener

```
criticalHitBtn.addEventListener('click', () => {
  criticalHitDiv.style.display = 'block';
  fumbleDiv.style.display = 'none';
});
```

Fumble Button Listener

```
fumbleBtn.addEventListener('click', () => {
  fumbleDiv.style.display = 'block';
  criticalHitDiv.style.display = 'none';
});
```

Event Listeners for Roll Generation and Submission

These listeners handle generating and processing roll values for both critical hits and fumbles. The code for both sections is similar.

Random Roll Button Listener

This listener generates a random roll.

```
criticalHitRandomBtn.addEventListener('click', () => {
  const roll = Math.floor(Math.random() * 100) + 1;
  criticalHitInput.value = roll;
  displayCriticalHitResult(roll);
});
```

Explanation
When the button is clicked, a random integer between 1 and 100 is generated and put into the input field. The system then calls the corresponding display function to process and show the result.

Submit Button Listener

This listener processes rolls from the input field.

```
criticalHitSubmitBtn.addEventListener('click', () => {
  const roll = parseInt(criticalHitInput.value);
  if (roll >= 1 && roll <= 100) {
    displayCriticalHitResult(roll);
  } else {
    criticalHitResultDiv.textContent = "Please enter a number between 1 and 100.";
  }
});
```

Explanation
When clicked, the code converts the input value to an integer. It validates the roll is between 1 and 100. If valid, it calls the display function. Otherwise, it shows an error message.

Utility Function: findResult

This function finds an entry in a data table that matches a given roll value. It can handle both single numbers and ranges.

```
function findResult(table, roll) {
  for (let i = 0; i < table.length; i++) {
    const range = table[i]["% Roll"].split("-");
    if (range.length === 1) {
      if (parseInt(range[0]) === roll) {
        return table[i];
      }
    } else if (range.length === 2) {
      const min = parseInt(range[0]);
      const max = parseInt(range[1]);
      if (roll >= min && roll <= max) {
        return table[i];
      }
    }
  }
  return null;
}
```

Explanation
The function iterates through a table, checking the "% Roll" property of each entry. It returns the matching entry if found, otherwise it returns null.

Display Functions

These functions format and display the results for critical hit and fumble rolls.

displayCriticalHitResult Function

This function displays the result for a critical hit.

```
function displayCriticalHitResult(roll) {
  const result = findResult(criticalHitTable, roll);
  if (result) {
    let description = `Roll: ${roll}\n`;
    for (const key in result) {
      if (key !== "% Roll") {
        description += `${key}: ${result[key]}\n`;
      }
    }
    criticalHitResultDiv.innerHTML = description;
  } else {
    criticalHitResultDiv.textContent = "No result found.";
  }
}
```

displayFumbleResult Function

This function displays the result for a fumble.

```
function displayFumbleResult(roll) {
  const result = findResult(fumbleTable, roll);
  if (result) {
    let description = `Roll: ${roll}\n`;
    for (const key in result) {
      if (key !== "% Roll") {
        description += `${key}: ${result[key]}\n`;
      }
    }
    fumbleResultDiv.innerHTML = description;
  } else {
    fumbleResultDiv.textContent = "No result found.";
  }
}
```

Explanation
Both functions work the same way. They accept a roll, look it up in their respective tables (critical or fumble) using findResult(), and then display the formatted result. If no match is found, they show a "No result found" message.