Etch-a-sketch Documentation

1. Description

The Etch-A-Sketch game is a digital representation of the classic drawing toy, and it is designed as part of the web development course of The Odin Project. The interface allows users to "draw" on the screen using the cursor inside a grid canvas.

It is created using HTML, CSS, and JavaScript. The project served me as an introduction to the world of web applications. HTML displays the structural foundation of the game's interface, CSS adds stylistic elements to improve the visual experience while JavaScript brings the game to life by responding to user inputs.

The interface is clean and user-friendly, with controls to clear the canvas, enable a rainbow mode for colorful strokes, and adjust the canvas size for more or less detailed drawings.

2. Objectives

The Etch-A-Sketch game project had several key objectives:

- Demonstrate a foundational knowledge of HTML, CSS, and JavaScript by constructing a interactive web application.
- Provide an intuitive user experience with clear instructions and functionality.
- Ensure responsive feedback to user actions, allowing real-time interaction with the canvas.
- Incorporate essential web development concepts such as DOM manipulation, event handling, and dynamic styling.

3. Design and Code

The ‚index.html' file serves as the foundation of the Etch-a-Sketch game. It is one of the three main files, alongside ‚index.js' for functionality and ‚style.css' for presentation. Together they make up the structure and behavior of the game.

In the ‚index.html' file, we have the following main components:

Head Section:

- Meta Tags: These tags ensure proper rendering and compatibility with various browsers and devices.
- Font Links: Connections to Google Fonts are established here for custom typing in the game interface.
- CSS Link: The external stylesheet style.css is linked here to apply styles to the game.
- JavaScript Link: The ‚index.js' file is included with the defer attribute, which ensures that the script executes after the document has been parsed.

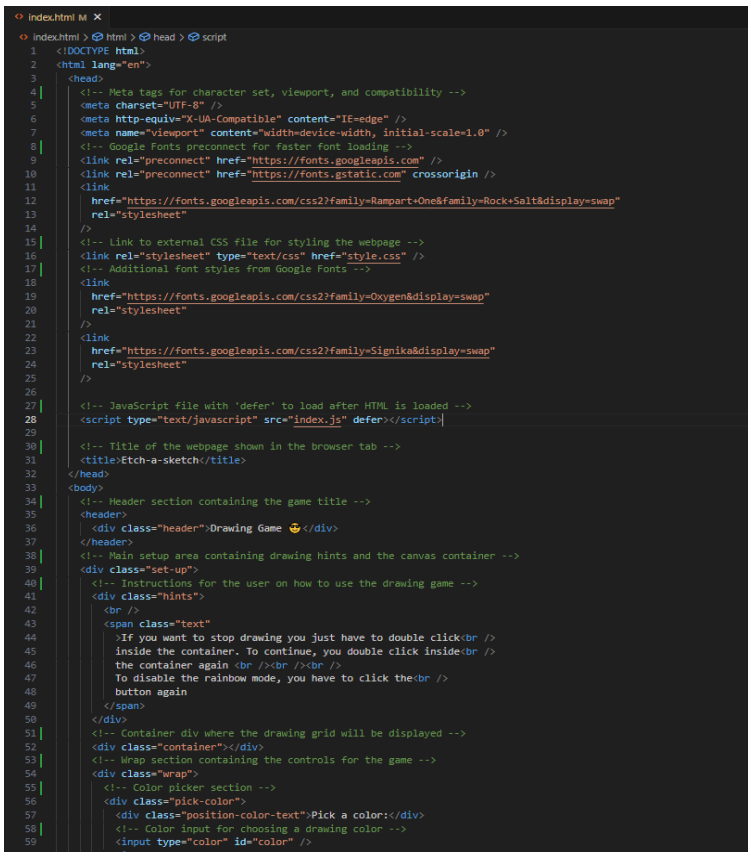
Body Section:

- Header: Contains the title of the game ‚Drawing Game 😎', which is displayed at the top of the page.
- Setup Area: This is the interactive part of the game where users can:
  - Read instructions on how to operate the game, such as stopping and continuing the drawing and toggling the rainbow mode.
  - Interact with the color picker to choose a drawing color.
  - Use buttons to clear the drawing area or toggle the rainbow drawing mode.
  - Adjust the canvas size using a range slider, with the current size displayed alongside.
- Canvas Container: An empty div that will later be populated with the drawing grid through JavaScript.

Footer section:

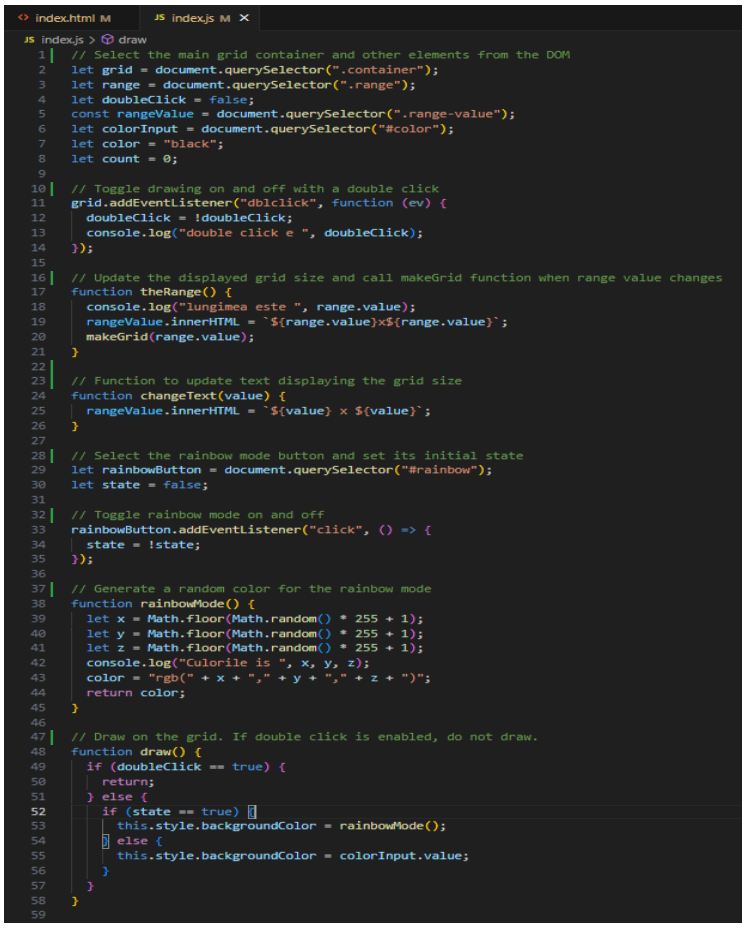
- A section that displays my name.

This is how the ‚index.html' file looks like:



```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Meta tags for character set, viewport, and compatibility -->
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <!-- Google Fonts preconnect for faster font loading -->
    <link rel="preconnect" href="https://fonts.googleapis.com" />
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />
    <link
      href="https://fonts.googleapis.com/css2?family=Rampart+One&family=Rock+Salt&display=swap"
      rel="stylesheet"
    />
    <!-- Link to external CSS file for styling the webpage -->
    <link rel="stylesheet" type="text/css" href="style.css" />
    <!-- Additional font styles from Google Fonts -->
    <link
      href="https://fonts.googleapis.com/css2?family=Oxygen&display=swap"
      rel="stylesheet"
    />
    <link
      href="https://fonts.googleapis.com/css2?family=Signika&display=swap"
      rel="stylesheet"
    />

    <!-- JavaScript file with 'defer' to load after HTML is loaded -->
    <script type="text/javascript" src="index.js" defer></script>

    <!-- Title of the webpage shown in the browser tab -->
    <title>Etch-a-sketch</title>
  </head>
  <body>
    <!-- Header section containing the game title -->
    <header>
      <div class="header">Drawing Game 🤠</div>
    </header>
    <!-- Main setup area containing drawing hints and the canvas container -->
    <div class="set-up">
      <!-- Instructions for the user on how to use the drawing game -->
      <div class="hints">
        <br />
        <span class="text"
          >If you want to stop drawing you just have to double click<br />
          inside the container. To continue, you double click inside<br />
          the container again <br /><br /><br />
          To disable the rainbow mode, you have to click the<br />
          button again
        </span>
      </div>
      <!-- Container div where the drawing grid will be displayed -->
      <div class="container"></div>
      <!-- Wrap section containing the controls for the game -->
      <div class="wrap">
        <!-- Color picker section -->
        <div class="pick-color">
          <div class="position-color-text">Pick a color:</div>
          <!-- Color input for choosing a drawing color -->
          <input type="color" id="color" />
        </div>

        <!-- Clear button to reset the drawing area -->
        <div class="clear-button">
          <button class="clear" onclick="clearAndRefresh()">Clear</button>
        </div>
        <!-- Button to toggle rainbow drawing mode -->
        <div class="rainbow-button">
          <button id="rainbow" onclick="rainbowMode()">Rainbow Mode</button>
        </div>
        <!-- Canvas size control -->
        <div class="canvas-size">
          <span class="canvas-font">Canvas size:</span>
          <!-- Range input to adjust the size of the drawing grid -->
          <input
            class="range"
            type="range"
            id="range"
            onclick="theRange()"
            min="4"
            max="100"
            value="16"
          />
          <!-- Display the current grid size -->
          <span class="range-value">16x16</span>
        </div>
      </div>
    </div>
    <!-- Footer section -->
    <footer>
      <div class="subsol">
        <div class="subsol-text">
          <center>Created for TOP by Szőke-Manea Alexandru</center>
        </div>
      </div>
    </footer>
  </body>
</html>
```

Moving on, we have the ‚index.js' which contains event listeners and functions that handle user interactions, such as drawing and changing colors. Here is how the code looks like:



```javascript
// Select the main grid container and other elements from the DOM
let grid = document.querySelector(".container");
let range = document.querySelector(".range");
let doubleClick = false;
const rangeValue = document.querySelector(".range-value");
let colorInput = document.querySelector("#color");
let color = "black";
let count = 0;

// Toggle drawing on and off with a double click
grid.addEventListener("dblclick", function (ev) {
  doubleClick = !doubleClick;
  console.log("double click e ", doubleClick);
});

// Update the displayed grid size and call makeGrid function when range value changes
function theRange() {
  console.log("lungimea este ", range.value);
  rangeValue.innerHTML = `${range.value}x${range.value}`;
  makeGrid(range.value);
}

// Function to update text displaying the grid size
function changeText(value) {
  rangeValue.innerHTML = `${value} x ${value}`;
}

// Select the rainbow mode button and set its initial state
let rainbowButton = document.querySelector("#rainbow");
let state = false;

// Toggle rainbow mode on and off
rainbowButton.addEventListener("click", () => {
  state = !state;
});

// Generate a random color for the rainbow mode
function rainbowMode() {
  let x = Math.floor(Math.random() * 255 + 1);
  let y = Math.floor(Math.random() * 255 + 1);
  let z = Math.floor(Math.random() * 255 + 1);
  console.log("Culorile is ", x, y, z);
  color = "rgb(" + x + "," + y + "," + z + ")";
  return color;
}

// Draw on the grid. If double click is enabled, do not draw.
function draw() {
  if (doubleClick == true) {
    return;
  } else {
    if (state == true) {
      this.style.backgroundColor = rainbowMode();
    } else {
      this.style.backgroundColor = colorInput.value;
    }
  }
}
```

```javascript
  }
}

// Update the drawing color when a new color is selected
colorInput.addEventListener("input", () => {
  color = colorInput.value;
  console.log("culoarea e ", color);
});

// Clear the drawing from the grid
function clearCanvas() {
  let elements = document.querySelectorAll(".cell");
  for (let i = 0; i < elements.length; i++) {
    elements[i].remove();
  }
}

// Clear the canvas and reset the game
function clearAndRefresh() {
  clearCanvas();
  location.reload();
  range.value = 16;
  makeGrid(16);
}

// Create a new grid based on the selected size
function makeGrid(gridSize) {
  clearCanvas();
  grid.style.cssText = `grid-template-columns: repeat(${gridSize}, 1fr);
                        grid-template-rows: repeat(${gridSize}, 1fr);`;
  for (let i = 0; i < gridSize * gridSize; i++) {
    const cell = document.createElement("div");
    cell.classList.add("cell");
    cell.addEventListener("mouseover", draw);
    grid.appendChild(cell);
  }
}
makeGrid(16);
```

Now, let's explore the specifics of the index.js functions, the grid creation, and state management.

Functions and Grid Creation

,makeGrid(gridSize)' function:

The ,makeGrid' function is responsible for dynamically creating the drawing grid. It takes a single parameter, ,gridSize', which represents both the number of cells horizontally and vertically since the grid is square.

First, it clears any existing cells from the grid to start fresh. Then it sets the CSS for the grid container to define the number of columns and rows based on ,gridSize'.
After that it loops ,gridSize' squared times (to cover the entire grid area) to create individual ,cell' div elements. Each cell gets a ,mouseover' event listener that calls the draw function when the mouse passes over it.
These cells are appended to the grid container, creating a visual grid for the user to interact with.

```
82  // Create a new grid based on the selected size
83  function makeGrid(gridSize) {
84    clearCanvas();
85    grid.style.cssText = `grid-template-columns: repeat(${gridSize}, 1fr);
86                          grid-template-rows: repeat(${gridSize}, 1fr);`;
87    for (let i = 0; i < gridSize * gridSize; i++) {
88      const cell = document.createElement("div");
89      cell.classList.add("cell");
90      cell.addEventListener("mouseover", draw);
91      grid.appendChild(cell);
92    }
93  }
94  makeGrid(16);
95
```

,clearCanvas()' and ,clearAndRefresh()' functions:

,clearCanvas' finds all elements with the class "cell" and removes them from the DOM. This is used to clear the grid. ,clearAndRefresh' calls ,clearCanvas' , then reloads the page and resets the range value to 16, effectively resetting the game.

```
66  // Clear the drawing from the grid
67  function clearCanvas() {
68    let elements = document.querySelectorAll(".cell");
69    for (let i = 0; i < elements.length; i++) {
70      elements[i].remove();
71    }
72  }
73
74  // Clear the canvas and reset the game
75  function clearAndRefresh() {
76    clearCanvas();
77    location.reload();
78    range.value = 16;
79    makeGrid(16);
80  }
```

,draw()' function:

This function changes the background color of the cell divs that are hovered over by the mouse. If ,doubleClick' is true, drawing is disabled. If the state is ,true' , it applies the color returned by ,rainbowMode'. Otherwise, it sets the cell's color to the value selected by the color picker.

```
// Draw on the grid. If double click is enabled, do not draw.
function draw() {
  if (doubleClick == true) {
    return;
  } else {
    if (state == true) {
      this.style.backgroundColor = rainbowMode();
    } else {
      this.style.backgroundColor = colorInput.value;
    }
  }
}

```

‚rainbowMode()' function:

‚rainbowMode' generates a random color by selecting random values for red, green, and blue components. The +1 ensures that the value is never 0 because ‚Math.random()' can return a value from 0 up to but not including 1). The resulting RGB values range from 1 to 255, creating a varied color for the rainbow effect.

```
// Generate a random color for the rainbow mode
function rainbowMode() {
  let x = Math.floor(Math.random() * 255 + 1);
  let y = Math.floor(Math.random() * 255 + 1);
  let z = Math.floor(Math.random() * 255 + 1);
  console.log("Culorile is ", x, y, z);
  color = "rgb(" + x + "," + y + "," + z + ")";
  return color;
}

```

State Management

Double Click State: The ‚doubleClick' variable is a boolean that tracks whether drawing is enabled or disabled. It is toggled on a double-click event on the grid, allowing the user to enable or disable drawing as they wish.

```
// Select the main grid container and other elements from the DOM
let grid = document.querySelector(".container");
let range = document.querySelector(".range");
let doubleClick = false;
const rangeValue = document.querySelector(".range-value");
let colorInput = document.querySelector("#color");
let color = "black";
let count = 0;

// Toggle drawing on and off with a double click
grid.addEventListener("dblclick", function (ev) {
  doubleClick = !doubleClick;
  console.log("double click e ", doubleClick);
});
```

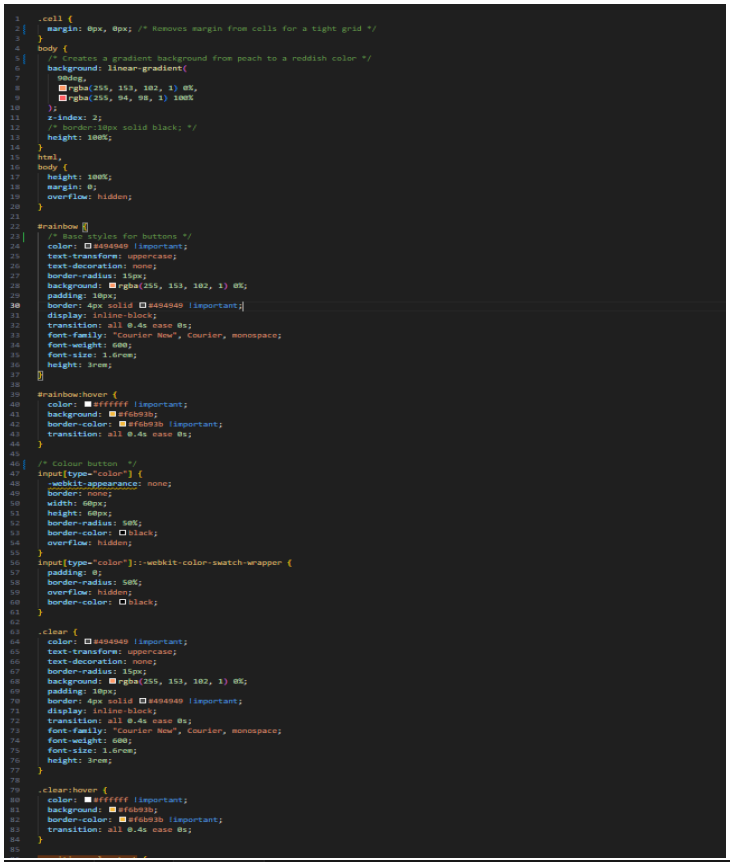
Rainbow Mode State:

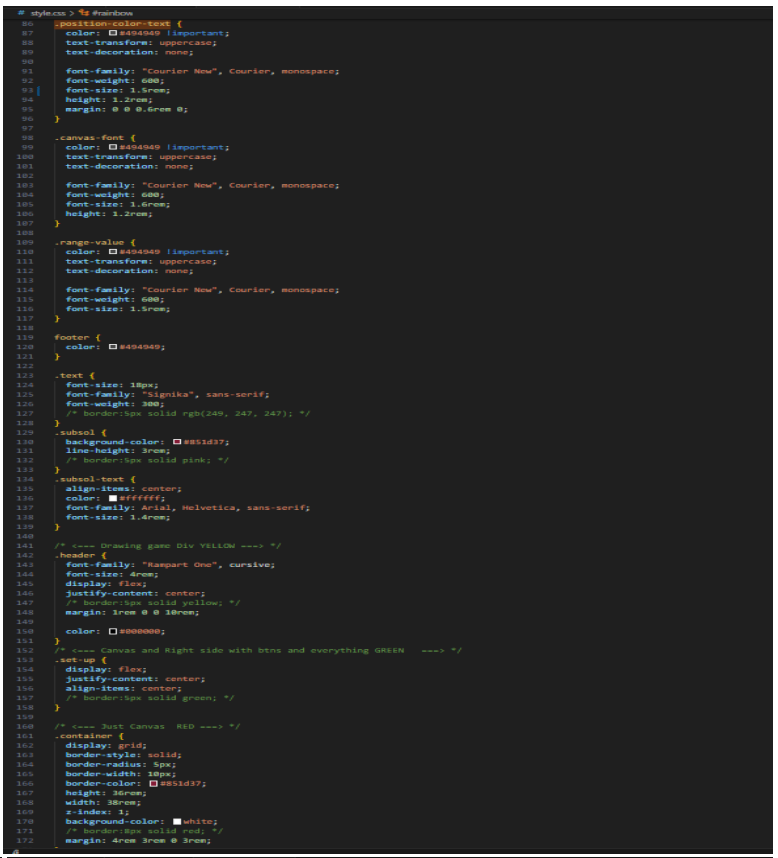
The state variable tracks whether the rainbow mode is active. It is toggled by the 'click' event on the rainbow button. If true, the draw function uses the rainbowMode to set the cell color. If it's ‚false', it uses the selected color from the color input.

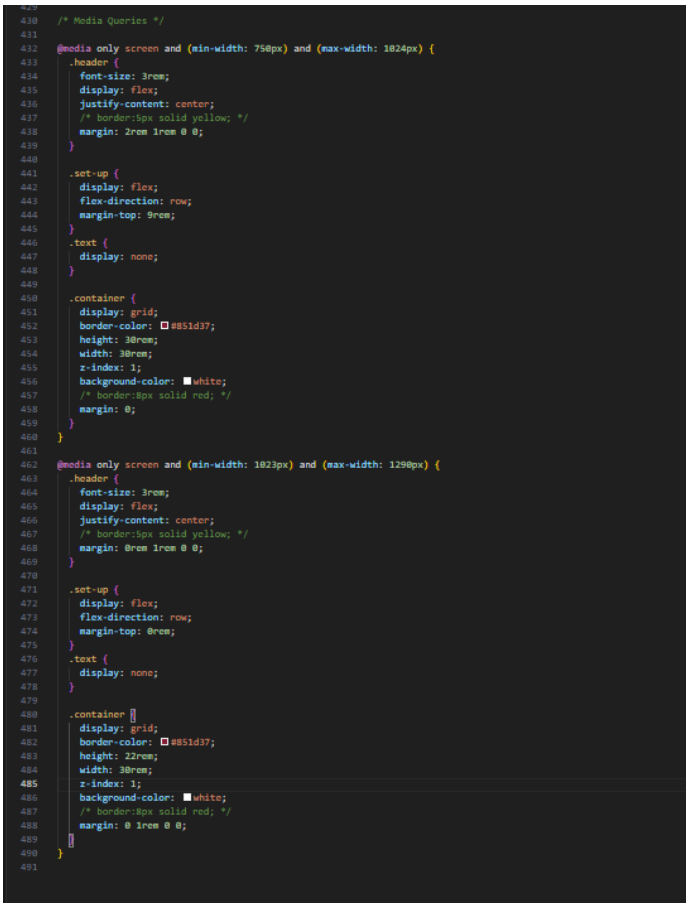
```
// Select the rainbow mode button and set its initial state
let rainbowButton = document.querySelector("#rainbow");
let state = false;

// Toggle rainbow mode on and off
rainbowButton.addEventListener("click", () => {
  state = !state;
});

```

By keeping track of these states the ‚index.js' controls the behavior of the drawing, allowing the user to toggle between different modes and choose when to draw on the grid.

Lastly we have the ‚style.css' file which contains the layout and structure for the game, it looks like this:







```
/* Media Queries */

@media only screen and (min-width: 750px) and (max-width: 1024px) {
  .header {
    font-size: 3rem;
    display: flex;
    justify-content: center;
    /* border:5px solid yellow; */
    margin: 2rem 1rem 0 0;
  }

  .set-up {
    display: flex;
    flex-direction: row;
    margin-top: 9rem;
  }
  .text {
    display: none;
  }

  .container {
    display: grid;
    border-color: #851d37;
    height: 30rem;
    width: 30rem;
    z-index: 1;
    background-color: white;
    /* border:8px solid red; */
    margin: 0;
  }
}

@media only screen and (min-width: 1023px) and (max-width: 1290px) {
  .header {
    font-size: 3rem;
    display: flex;
    justify-content: center;
    /* border:5px solid yellow; */
    margin: 0rem 1rem 0 0;
  }

  .set-up {
    display: flex;
    flex-direction: row;
    margin-top: 0rem;
  }
  .text {
    display: none;
  }

  .container {
    display: grid;
    border-color: #851d37;
    height: 22rem;
    width: 30rem;
    z-index: 1;
    background-color: white;
    /* border:8px solid red; */
    margin: 0 1rem 0 0;
  }
}
```

General Styles

The .cell class gives basic styling for the individual cells within the grid. The body sets up a gradient background for the entire page, making use of CSS3 gradients. The html, body selector ensures the full height of the page is utilized and no default margin is applied.

Interactive Elements

Styles for interactive elements like buttons (#rainbow, .clear) and inputs (input[type="color"]) are defined here, ensuring that these elements are visually appealing and provide feedback on interaction (like hover effects).

Text and Positioning

The styles for text elements (.position-color-text, .canvas-font, .range-value) set the font properties and uppercasing to ensure consistency and readability across the game's interface.

Layout and Structure

The .container, .wrap, .header, and footer selectors are responsible for the layout of the game's container, header, and footer sections. Flexbox is used in .set-up to center the content and to align items in the middle.

Responsive Design

Media queries are used to adjust the styling for different viewport widths, ensuring the game is presentable and functional on various device sizes. For instance, font sizes and the layout of elements are adjusted to fit smaller screens.

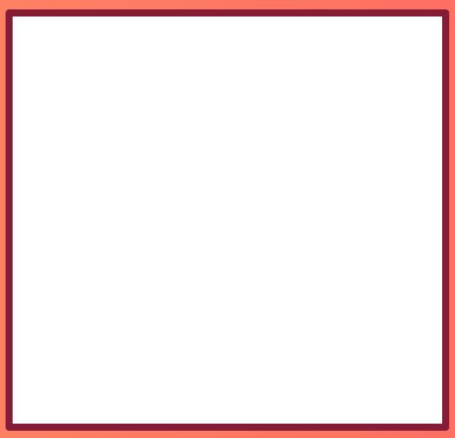
The CSS file for the game is quite elaborate, with careful attention to responsive design, interactive feedback, and a visually appealing theme.

The final product looks like this:

Drawing Game 😎

If you want to stop drawing you just have to double click inside the container. To continue, you double click inside the container again

To disable the rainbow mode, you have to click the button again



PICK A COLOR:

CLEAR

RAINBOW MODE

CANVAS SIZE: 16X16