

Hands-on Lab: Develop Colorful Memory Match Game Using JavaScript DOM



Estimated time needed: 30 minutes

What you will learn

In this lab, you will learn the fundamentals of building a memory matching game using HTML and JavaScript. You will understand how to dynamically generate game elements, such as cards with various colors, handle click events to reveal and match colors, implement a basic scoring system that increments upon successful matches, create a timer mechanism to limit game time and initiate game restart functionalities. Through this lab, you will grasp key concepts of DOM manipulation, event handling, array manipulation (shuffling), and basic game logic, offering practical insight into creating interactive web-based games.

Learning objectives

After completing this lab, you will be able to:

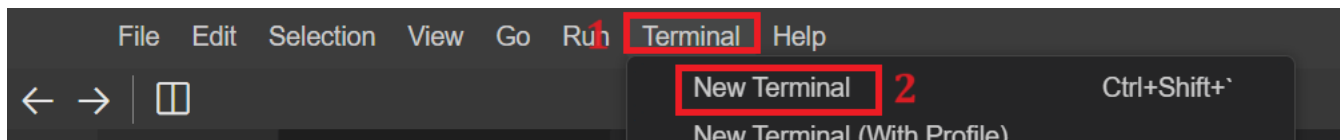
- **DOM manipulation:** Understand and implement dynamic HTML element creation and modification using JavaScript to generate a playable memory matching game grid.
- **Event handling:** Learn to manage and respond to user interactions by handling click events on game cards, revealing colors, and implementing logic for matching pairs.
- **Game logic implementation:** Develop fundamental game logic by incorporating mechanisms to match pairs of colors, track scores, reset the game, and manage game time through a simple timer.
- **Fundamentals of web game development:** Gain insights into core concepts essential for creating interactive web-based games, including array manipulation for shuffling elements, styling elements with CSS, and integrating JavaScript functionalities for game interactivity and dynamics.

Prerequisites

- Basic knowledge of HTML and GitHub.
- Web browser with a console (Chrome DevTools, Firefox Console, and so on).

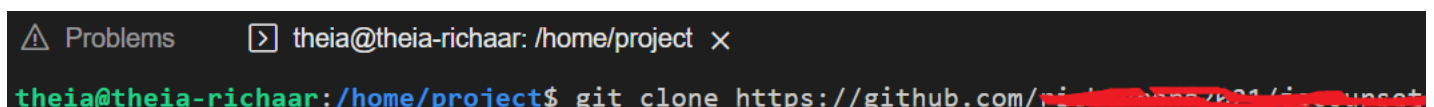
Step 1: Setting up the environment

1. Firstly, you need to clone your main repository in **Skills Network Environment** which you have created in the first lab and where you have pushed all of your previous labs files and folders. Follow given steps:
 - Click on terminal in top-right window pane and then select **New Terminal**.

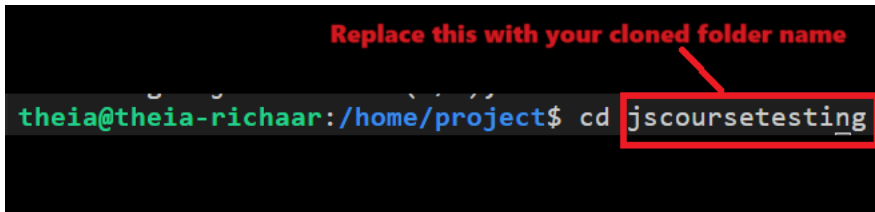


- Perform `git clone` command by writing given command in the terminal.
`git clone <github-repository-url>`

Note: Put your own GitHub repository link instead of `<github-repository-url>`.



- Above step will clone folder for your GitHub repository under project folder in explorer. You will also see multiple folders inside cloned folder.
- Now you need to navigate inside the cloned folder. For this write given command in the terminal:
`cd <repository-folder-name>`



Note: Write your cloned folder name instead of <repository-folder-name> which you have created in first lab. Perform `git clone` if you have logged out of **Skills Network Environment** and you cannot see any files or folder after you logged in.

- Now select **cloned Folder Name** folder, right click on it and click on **New Folder**. Enter folder name as **colorfulMemoryGame**. It will create the folder for you. Then select **colorfulMemoryGame** folder, right click and select **New File**. Enter file named as **colorful_memory_game.html** and click OK. It will create your HTML file.
- Now, select the **colorfulMemoryGame** folder again, right click and select **New File**. Enter the file named **colorful_memory_game.js** and click OK. It will create your JavaScript file.
- Again, select the **colorfulMemoryGame** folder, right click and select **New File**. Enter the file named **colorfulGame.css** and click OK. It will create your CSS file.
- Now, click on this link [colorfulGame.css](#) and copy the code from this link and paste it in **colorfulGame.css** file.

Step 2: Create HTML template structure

- Create a template structure of an HTML file by adding the following content:

```
<!DOCTYPE html>
<html>
<head>
  <title>Colorful Memory Match Game</title>
  <link rel="stylesheet" href="./colorfulGame.css">
</head>
<body>
  <h1>Colorful Memory Match Game</h1>
  <div class="container">
    <div id="game-container">
      <!-- Cards will be generated dynamically using JavaScript -->
    </div>
    <div class="startMain">
      <p id="score">Score: 0</p>
      <p id="timer">Time Left: 30</p>
      <button id="startbtn">Start/Restart</button>
    </div>
  </div>
  <script src="./colorful_memory_game.js"></script>
</body>
</html>
```

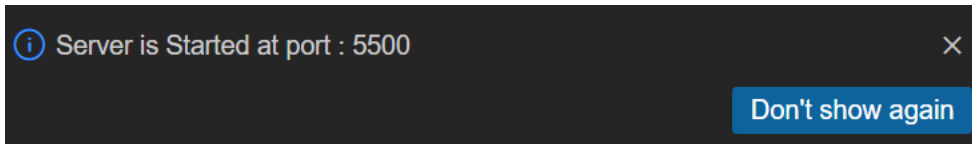
- The above HTML code includes the following:


- The structure for a colorful memory match game including a title, "Colorful Memory Match Game," and a game elements container.
- It has link tag to include CSS file in the HTML file.
- Inside the container, there's a specific division <div> intended to hold dynamically generated cards using JavaScript, although the card generation code isn't present in this snippet.
- Additionally, there's a section below the game container with a paragraph displaying the current score and a timer indicating the time left, initially set to 0 and 30 seconds, respectively. Finally, there's a button labeled "Start/Restart" <button> intended to initiate or reset the game when clicked.
- To include the JavaScript file in **colorful_memory_game.html**, a script tag can be used above the </body> tag.

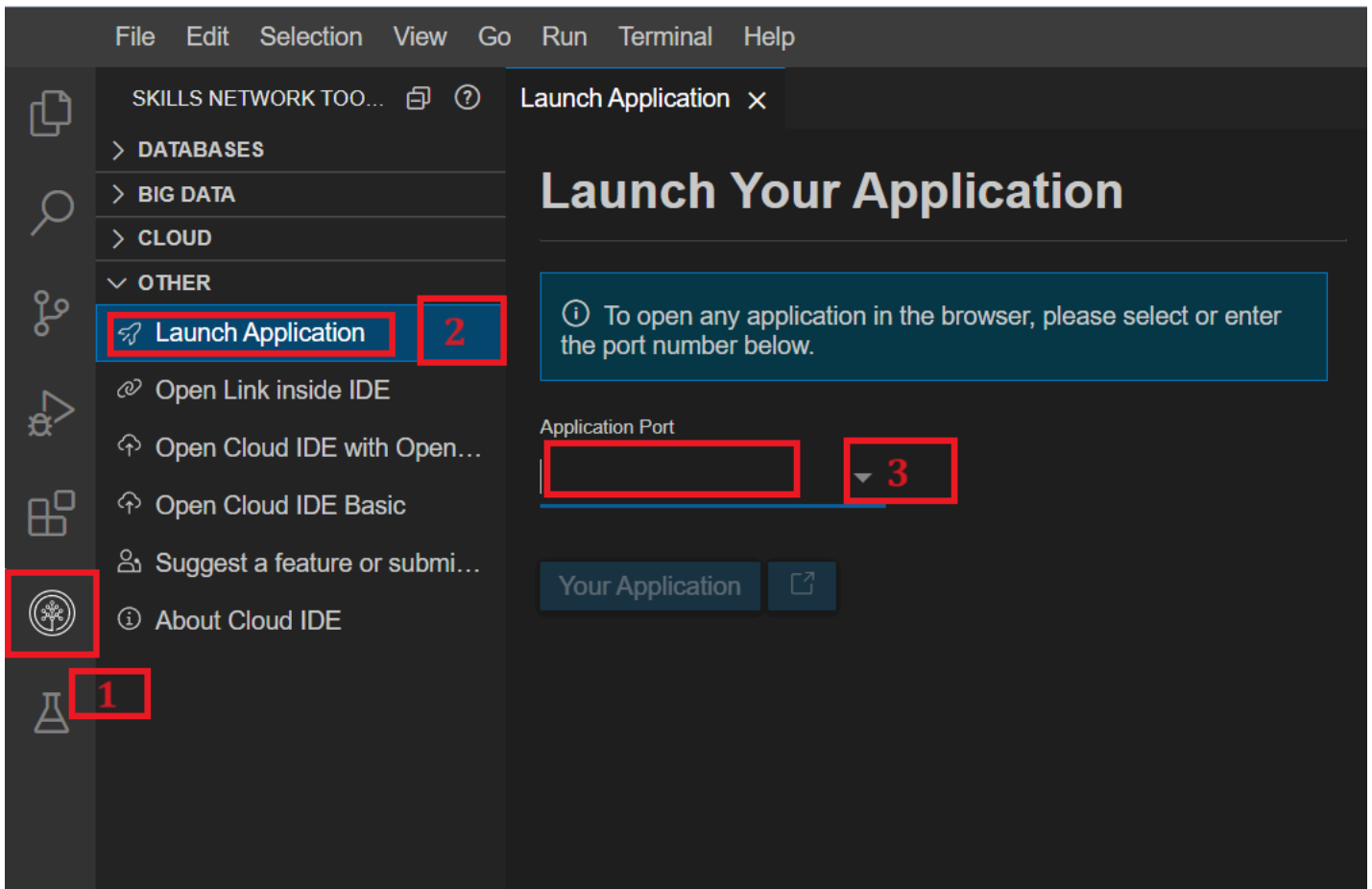
Note: When you have pasted the code, save your file.

Step 3: Check the output

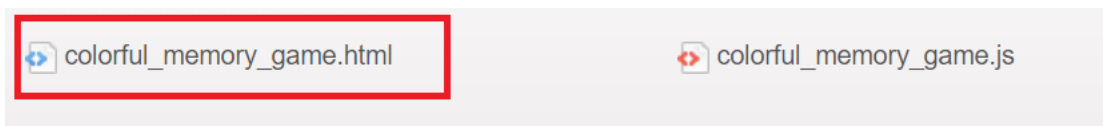
1. To view how your HTML page will be displayed in the browser, use the built-in Live Server extension. Select file **colorful_memory_game.html** within **colorfulMemoryGame** folder and right-click on that file and choose 'Open with Live Server'.
2. A notification will appear at the bottom-right, indicating that the server has started on port 5500.



3. Click the Skills Network button on the left (refer to number 1) to open the "Skills Network Toolbox". Then select Launch Application (refer to number 2). From there, enter port 5500 at number 3 and click this button .



4. It will open your default browser where you will see **cloned-folder-name** folder name. Click on that **cloned-folder-name** folder name. After clicking you will see multiple folders name, among those folders, click on the **calculateArea** folder. You will see files related to this folder where again you will click on **calculate_Area.html** file as shown below.



5. It will show the output like below.

Colorful Memory Match Game

Score: 0

Time Left: 30

Start/Restart

Step 4: Perform Git commands

1. Perform `git add` to add the latest files and folder in the git environment.

```
git add --a
```

- Make sure the terminal has the path as follows:



2. Then perform `git commit` in the terminal. While performing `git commit`, terminal can show message to set up your `git config --global` for `user.name` and `user.email`. If yes, then you need to perform `git config` command as well for `user.name` and `user.email` as given.

```
git config --global user.email "you@example.com"
```

```
git config --global user.name "Your Name"
```

After this, perform git commit as given:

```
git commit -m "message"
```

3. Then, perform git push by writing given command in terminal.

```
git push origin
```

- After the push command, the system will prompt you to enter your username and password. Enter the username for your GitHub account and the password that you created in the first lab. After entering the credentials, all of your latest folders and files will be pushed to your GitHub repository.

Note: Repeat step get add, git commit, and git push every time you made change in the code while working in **Skills Network Environment**.

Step 5: Defining variables to access data

1. Initialization of arrays and variables using given code:

- Include the given code in javaScript file.

```
const colors = ['red', 'blue', 'green', 'purple', 'orange', 'pink', 'red', 'blue', 'green', 'purple', 'orange', 'pink'];
let cards = shuffle(colors.concat(colors));
let selectedCards = [];
let score = 0;
let timeLeft = 30;
let gameInterval;
```

- **colors array:** This array holds distinct color values in strings, representing the colors for the cards in the memory match game. These colors create pairs for the game.
- **cards array:** Initialized by shuffling and attaching the 'colors' array, this 'cards' array holds the color values for the cards in the game. The shuffle function employs the Fisher-Yates algorithm to randomize the order of the colors and then duplicates these colors to create pairs, forming the set of cards for gameplay.
- **selectedCards:** This variable acts as a temporary storage for the currently selected cards during the game. When a player clicks on a card, it gets added to this array to enable match comparisons.
- **score:** This variable tracks the player's score throughout the game. The score gets incremented whenever the player matches a pair of cards successfully. It's updated and displayed to reflect the player's progress and performance.
- **timeLeft:** It represents the time remaining for the player to complete the game. Initially set to a specific duration, it counts down as the game progresses. When it reaches zero, the game ends.
- **gameInterval:** This variable manages the game timer. It's utilized to control the countdown mechanism for the game's duration. The interval continuously decrements the 'timeLeft' variable, updating the displayed time and triggering the game's end when the time expires.

2. DOM element selection:

- For this, include the given code after the previous code.

```
const startbtn = document.getElementById('startbtn');
const gameContainer = document.getElementById('game-container');
const scoreElement = document.getElementById('score');
const timerElement = document.getElementById('timer');
```

- **startbtn:** This variable is assigned the HTML element with the ID 'startbtn'. It typically represents a button element intended to start or restart the game when clicked. This variable allows the JavaScript code to access and manipulate this specific button element.
- **gameContainer:** This variable is assigned the HTML element with the ID 'game-container'; it refers to a div or container element that dynamically generates cards for the memory match game. It allows JavaScript to manipulate or append child elements (cards) within this container.
- **scoreElement:** This variable represents the HTML element with the ID 'score'. It is associated with a paragraph or span element displaying the player's score during the game. JavaScript can update the displayed score by manipulating this specific element's content.
- **timerElement:** This variable refers to the HTML element identified by the ID 'timer'. It's presumably linked to a paragraph or span element that displays the time remaining for the player to complete the game. JavaScript can update this element to reflect the countdown and notify the player about the remaining time.

Step 6: Create and call functions to start the game

1. Create the **generateCards()** function responsible for dynamically creating the card elements within the game container based on the 'cards' array that holds color values for the cards. This function creates the card elements dynamically within the game-container div. Include given code in JavaScript file after previous code.

```
function generateCards() {
  for (const color of cards) {
    const card = document.createElement('div');
    card.classList.add('card');
    card.dataset.color = color;
    card.textContent = '?';
    gameContainer.appendChild(card);
  }
}
```

- It utilizes a 'for...of' loop to iterate over each element (color) in the 'cards' array. For each color in the 'cards' array:
 - Inside the loop, it creates a new HTML div element using `document.createElement('div')`. This 'div' element represents a card in the game.
 - It adds a class 'card' to the newly created 'div' element using `card.classList.add('card')`. This class might contain CSS styles or rules to style the card elements.
 - It sets the 'dataset.color' attribute of the card element to the current color value from the 'cards' array. This icon represents the card's hidden color until the player clicks it.
 - The text content of each card is initially set to a question mark ('?') using the `card.textContent = '?'`. This represents that the color of the card is hidden until it's clicked by the player.
 - Finally, the newly created card element is attached to the 'gameContainer' element as a child. This action adds each card element to the game interface within the designated container.
2. The **shuffle()** Function is responsible for shuffling the elements of an array in random order. It uses the Fisher-Yates shuffle algorithm, a common method for randomizing the order of elements in an array. Include given code after **generateCards()** function.

```
function shuffle(array) {
  for (let i = array.length - 1; i > 0; i--) {
    const j = Math.floor(Math.random() * (i + 1));
    [array[i], array[j]] = [array[j], array[i]];
  }
  return array;
}
```

- **Array parameter:** It takes an array as an argument, which contains yet to be shuffled elements.
 - **Shuffling process using loop through the array:** The function starts by initiating a 'for' loop that iterates backward through the array starting from the last index (let `i = array.length - 1; i > 0; i--`).
 - **Random index selection:** Within each iteration, it generates a random index 'j' using `Math.floor(Math.random() * (i + 1))`. This 'j' represents a random index within the array.
 - **Swapping elements:** It then swaps the elements at the 'i' and 'j' indices using array destructuring assignment: `[array[i], array[j]] = [array[j], array[i]]`. This line efficiently swaps the values at positions 'i' and 'j' without requiring a temporary variable.
 - **Continuing the loop:** The loop continues until it finishes iterating through the entire array, shuffling elements along the way.
 - **Returning the shuffled array:** Once the loop is complete, the function returns the array with its elements rearranged into a random order.
3. The **handleCardClick(event)** function manages the logic when a user clicks the card in the memory match game. Include given code after **shuffle()** function. Let's break down each step within this function:

```
function handleCardClick(event) {
  const card = event.target;
  if (!card.classList.contains('card') || card.classList.contains('matched')) {
    return;
  }
  card.textContent = card.dataset.color;
  card.style.backgroundColor = card.dataset.color;
  selectedCards.push(card);
  if (selectedCards.length === 2) {
    setTimeout(checkMatch, 500);
  }
}
```

- **Event Target using `const card = event.target`;** This line retrieves the element that triggered the event (in this case, a clicked card) and assigns it to the 'card' variable.
 - **Checking the card:** `if (!card.classList.contains('card') || card.classList.contains('matched')) { return; }` This 'if' statement checks whether the clicked element is a card and if it's already matched. If either condition is true:
 - If the element is not a card or has already matched, the function returns early, ignoring any further actions for this particular click.
 - **Revealing the card:**
 - `card.textContent = card.dataset.color`; It sets the text content of the clicked card to the value stored in its 'dataset.color'. This action reveals the card's color by changing the text content to the color value.
 - `card.style.backgroundColor = card.dataset.color`; Changes the card's background color to match the revealed color.
 - **Handling selected cards:**
 - `selectedCards.push(card)`; Adds the clicked card to the 'selectedCards' array, indicating that it's one of the cards currently chosen by the player.
 - **Checking for matches:**
 - `if (selectedCards.length === 2) { setTimeout(checkMatch, 500); }` Checks if two cards have been selected. If two cards have been chosen, it uses 'setTimeout()' to delay the execution of the 'checkMatch()' function by 500 milliseconds. This brief delay allows the player to see both selected cards before their comparison briefly.
4. The **checkMatch()** function evaluates whether the two selected cards match each other in the memory match game. Include given code after **handleCardClick()** function.

```
function checkMatch() {
  const [card1, card2] = selectedCards;
  if (card1.dataset.color === card2.dataset.color) {
    card1.classList.add('matched');
    card2.classList.add('matched');
    score += 2;
    scoreElement.textContent = `Score: ${score}`;
  } else {
    card1.textContent = '?';
    card2.textContent = '?';
    card1.style.backgroundColor = '#ddd';
    card2.style.backgroundColor = '#ddd';
  }
  selectedCards = [];
}
```

Here's a detailed breakdown of the function:

- Destructuring selected cards:
 - **const [card1, card2] = selectedCards;** This line uses array destructuring to assign the first two elements of the 'selectedCards' array to 'card1' and 'card2'. These variables represent the two cards selected by the player for comparison.
- Comparing card colors:
 - **if (card1.dataset.color === card2.dataset.color) { ... }:** This checks if the color value stored in the 'dataset.color' attribute of 'card1' matches the color value of 'card2'.
 - If the colors match: It adds the class 'matched' to both cards using **'classList.add('matched')'**, marking them as matched pairs in the game.
 - Increases the 'score' by 2 points, as the player successfully matched a pair.
 - Updates the 'scoreElement.textContent' to display the updated score to the player.
- Handling non-matching cards: If the colors of the two selected cards don't match, it resets the text content of both cards to a question mark ('?'), hiding their colors again.
 - Sets the background color of both cards to a default color ('#ddd'), providing a visual cue that the selected cards didn't match.
- Resetting selection:
 - **selectedCards = [];** It clears the 'selectedCards' array to reset it for the next set of card selections. This action ensures the player can select two new cards after the comparison.

5. The **startGame()** Function is a pivotal part of initializing and starting the memory match game. Include given code after **checkMatch()** function.

```
function startGame() {
  let timeLeft = 30;
  startbtn.disabled = true;
  score = 0; // Reset score to zero
  scoreElement.textContent = `Score: ${score}`;
  startGameTimer(timeLeft);
  cards = shuffle(colors.concat(colors));
  selectedCards = [];
  gameContainer.innerHTML = '';
  generateCards();
  gameContainer.addEventListener('click', handleCardClick);
}
```

Here's a summary of its functionalities:

- Setting initial game state:
 - **let timeLeft = 30;** Initializes the 'timeLeft' variable to 30 seconds, setting the duration for the game.
 - **startbtn.disabled = true;** Disables the 'startbtn' button to prevent multiple game initiations simultaneously, ensuring one game is in progress at a time.
 - **score = 0;** Resets the 'score' variable to zero, initializing it for the new game.
 - **scoreElement.textContent = `Score: \${score}`;** Updates the displayed score to show that it's reset to zero for the new game.
- Starting the game timer:
 - **startGameTimer(timeLeft);** Initiates the game timer, counting down from the specified 'timeLeft' duration.
- Preparing cards and game elements:
 - **cards = shuffle(colors.concat(colors));** Shuffles the 'colors' array and duplicates it to create pairs for the game cards.
 - **selectedCards = [];** Clears the 'selectedCards' array to prepare for new card selections in the upcoming game.
 - **gameContainer.innerHTML = '';** Clears the game container, removing any existing cards from previous games.
 - **generateCards();** Generates a new set of cards within the game container by calling the 'generateCards()' function, creating a fresh game layout for the player.
- Enabling card click event:

- `gameContainer.addEventListener('click', handleCardClick);` Adds an event listener to the game container, enabling card clicks and triggering the `'handleCardClick()'` function to manage the gameplay when cards are clicked.

6. The **startGameTimer(timeLeft)** function manages the game timer, updating the displayed time and handling the end of the game when the timer reaches zero. Include after **startGame() Function**.

```
function startGameTimer(timeLeft) {
  timerElement.textContent = `Time Left: ${timeLeft}`;
  gameInterval = setInterval(() => {
    timeLeft--;
    timerElement.textContent = `Time Left: ${timeLeft}`;
    if (timeLeft === 0) {
      clearInterval(gameInterval);
      let timeLeft = 30;
      alert('Game Over!');
      startbtn.disabled = false;
    }
  }, 1000);
}
```

Here's a detailed explanation of its workings:

- Initial display:
 - `timerElement.textContent = Time Left: ${timeLeft};` Sets the initial display of the timer to show the 'timeLeft' value, indicating the starting time remaining for the game.
 - Interval setup:
 - `gameInterval = setInterval(() => { ... }, 1000);` Initiates an interval that triggers a function every second (1000 milliseconds) to update the timer.
 - Countdown:
 - `timeLeft--;` Decrements the 'timeLeft' variable every second within the interval, simulating the countdown by reducing the remaining time.
 - Updating displayed time:
 - `timerElement.textContent = Time Left: ${timeLeft};` Updates the displayed time on the HTML element ('timerElement') to reflect the updated 'timeLeft' value after each decrement.
 - End of game:
 - `if (timeLeft === 0) { ... };` Checks if the remaining time reaches zero.
 - If 'timeLeft' equals zero:
 - `clearInterval(gameInterval);` Stops the interval, effectively ending the timer from counting down further.
 - `let timeLeft = 30;` This line is redundant as it re-declares 'timeLeft' within the scope of this block, resetting it to 30, but it does not affect the 'timeLeft' used in the interval.
 - `alert('Game Over!');` Displays an alert indicating that the game is over because the time limit has been reached.
 - `startbtn.disabled = false;` Re-enables the 'startbtn' button, allowing the player to start a new game after the current one has ended.
7. **Event listeners:** To listen, click event **startbtn** adds an event listener to the 'startbtn' element, triggering the 'startGame' function when the button is clicked. Include the given code at the end of javaScript file.

```
startbtn.addEventListener('click', startGame);
```

Step 7: Check the output

1. Again check the output. It will be as shown below.

Colorful Memory Match Game

?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?

Score: 0

Time Left: 30

Start/Restart

2. By clicking the **Start/Restart** button, you can start matching the colors! If the color matches, the score will update. The timer will also start as you click the **Start/Restart** button.

Colorful Memory Match Game

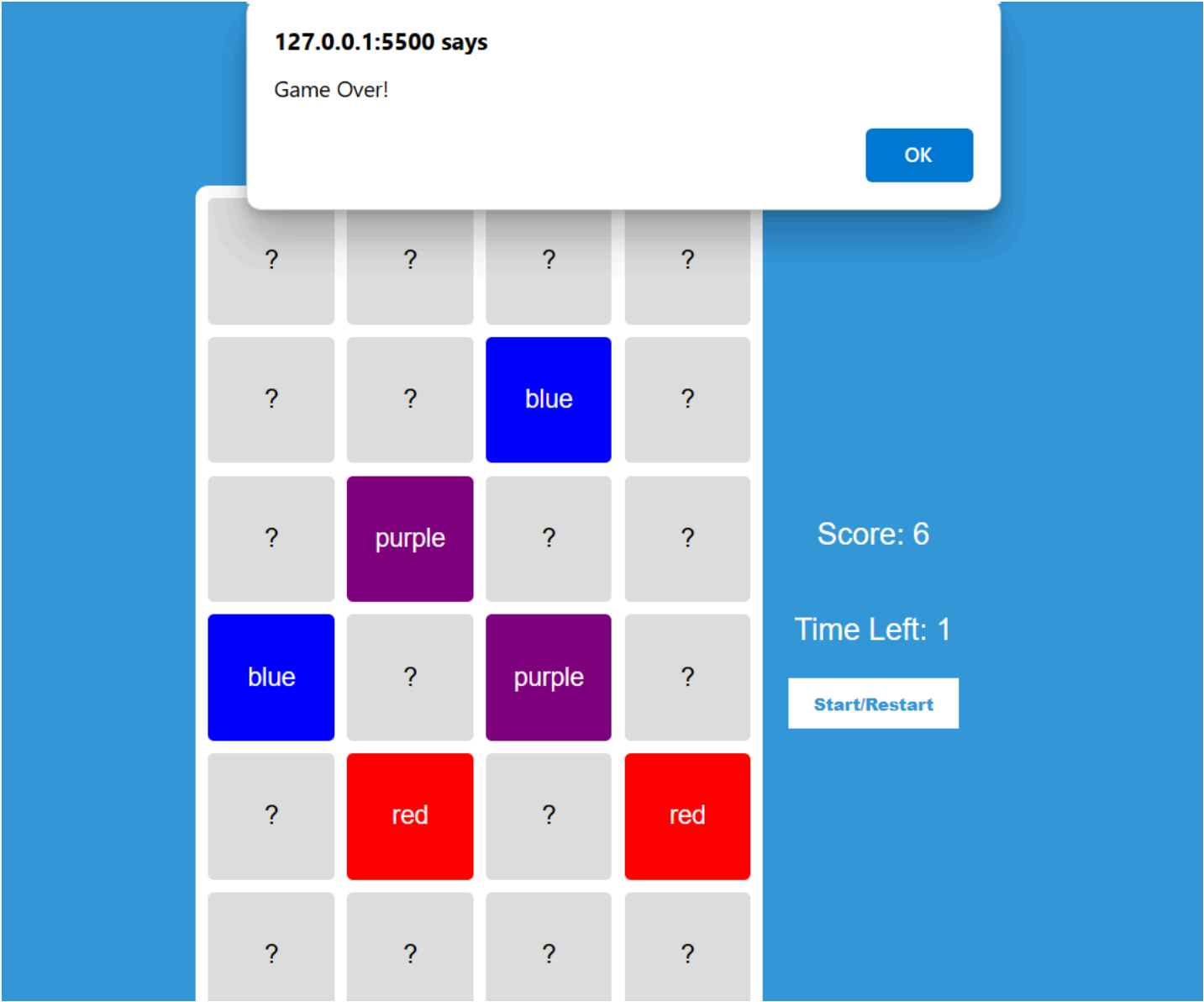
?	?	?	?
?	?	blue	?
?	purple	?	?
blue	?	purple	?
?	red	?	red
?	?	?	?

Score: 6

Time Left: 8

Start/Restart

3. When the time has ended, it will generate a popup box to alert **Game Over!**



5. Perform `git add`, `git commit`, and `git push` commands to update changes into your **colorfulMemoryGame** folder; GitHub repository for proper code management.

Summary

- 1. **HTML structure:** Defines a game titled "Colorful Memory Match Game." It includes elements for the game grid, score, timer, and a start/restart button.
- 2. **Styling:** Applies CSS to create a visually appealing layout with specific colors, fonts, and button styles.
- 3. **JavaScript functions:**
 - `generateCards()`: Creates card elements with colors for the game.
 - `shuffle(array)`: Randomizes the order of elements in an array.
 - `handleCardClick(event)`: Manages card clicks and checks for matches.
 - `startGame()`: Resets the game, shuffles cards, and starts a timer.
- 4. **Event listeners:** Listens for clicks on the start/restart button to initialize the game. Listens for card clicks to reveal colors and check for matching pairs.

© IBM Corporation 2023. All rights reserved.