



CSE341 - Internet Programming

Senior-1

Student name: Donia Sameh Farouk

Student ID: 20P3424

Introduction

The Card Matching Game is a classic memory challenge that tests players' ability to recall and match pairs of cards. The game typically involves a grid of face-down cards, each with a hidden image. Players flip two cards at a time, trying to find matching pairs. If a pair is found, the cards remain face-up; otherwise, they are flipped back down. The objective is to match all pairs in the shortest time possible, making it a fun and engaging way to exercise memory and concentration skills.

How the Game Works

1. **Start the Game:** The game begins with a welcome screen and start the game.
2. **Game Setup:** Once started, a grid of cards is displayed face-down. Each card has a matching pair hidden somewhere in the grid.
3. **Gameplay:**
 - The player clicks on two cards to flip them over.
 - If the images on the two flipped cards match, they get excluded from the board.
 - If they do not match, the cards are flipped back down after a short delay.
 - Each time a player flips two cards, it counts as one move.
 - Additional bonus points might be awarded for streaks of consecutive correct matches.
 - If user get a higher score , the score will be updated
4. **Winning the Game:** The game continues until all pairs are found and the board doesn't contain any card. The player's time and moves are tracked, and a message displays their performance.
5. **Resetting the Game:** Players can reset the game to end the game so he can play again with a new arrangement of cards.
6. **After winning or losing or resetting:** the user is asked to enter his username to update his name and information in the leaderboard.
7. **Leaderboard:** displays the usernames and information of the previous player

HTML CODE

This HTML structure provides a comprehensive setup for a card matching game, with elements for user interaction, game state display, sound effects, and leaderboard functionality.

The `head` section contains meta information about the HTML document, including the character set, viewport settings, the title, and a link to an external CSS file (`index.css`) for styling.

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```

<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Card Matching Game</title>
<link rel="stylesheet" href="index.css">
</head>

```

The body section:

Countdown and Welcome Message

- `countdown`: A div for displaying the countdown before the game starts.
- `welcome`: Contains the game's title, instructions on how to score points, and a prompt to start the game.

```

<body>
  <div class="countdown"></div>
  <div class="welcome">
    <h1>Card Matching Game</h1>
    <h2>Lets Play!<br>
      <span class="points">
        <ul><span style="font-weight: bold; text-decoration:
underline;">Score Points</span>:
          <li> 1 pair matched = 1 point</li>
          <li>2 consecutive pair matched = 2 points </li>
          <li>The points are incremented by 1 if you make
consecutive matches</li>
        </ul>
      </span>
    </h2>
    <h3>press button to start</h3>
  </div>

```

Game Information Display

- `info`: Contains the current state of the game, including the number of flips, the timer, and the score.
- `flips`: Displays the number of card flips made by the player.
- `timer`: Displays the remaining time.
- `score`: Displays the player's current score.

```

<div class="info">

```

```

    <div class="state">
      <div class="flips">Flips count: 0</div>
      <div class="timer">Timer: 60</div>
      <div class="score">Score: 0</div>
    </div>
  </div>

```

Game Container and Cards

- `game-container`: Contains the game elements, including sounds, cards, and messages.
- `audio`: Elements for playing different sound effects when cards are clicked, matched, or when the game ends.
- `all-cards`: A list of `li` elements, each representing a card with a front and back face. Each card can be clicked to reveal its front image.

Message is displayed when the player win or lose or reset the game

```

<div class="game-container">
  <audio id="clickSound">
    <source src="sounds/Audio_flip.wav">
  </audio>
  <audio id="matchSound">
    <source src="sounds/Audio_match.wav">
  </audio>
  <audio id="VictorySound">
    <source src="sounds/Audio_victory.wav">
  </audio>
  <audio id="gameOverSound">
    <source src="sounds/Audio_gameover.wav">
  </audio>
  <ul class="all-cards">
    <li class="card clickable">
      <div class="back card-shape">
        
      </div>
      <div class="front card-shape">
        
      </div>
    </li>
    <li class="card clickable">
      <div class="back card-shape">
        

```

```

        </div>
        <div class="front card-shape">
            
        </div>
    </li>
    <li class="card clickable">
        <div class="back card-shape">
            
        </div>
        <div class="front card-shape">
            
        </div>
    </li>
    <li class="card clickable">
        <div class="back card-shape">
            
        </div>
        <div class="front card-shape">
            
        </div>
    </li>
    <li class="card clickable">
        <div class="back card-shape">
            
        </div>
        <div class="front card-shape">
            
        </div>
    </li>
    <li class="card clickable">
        <div class="back card-shape">
            
        </div>
        <div class="front card-shape">
            
        </div>
    </li>
    <li class="card clickable">
        <div class="back card-shape">
            
        </div>
    </li>

```

```

        <div class="front card-shape">
            
        </div>
    </li>
    <li class="card clickable">
        <div class="back card-shape">
            
        </div>
        <div class="front card-shape">
            
        </div>
    </li>
    <li class="card clickable">
        <div class="back card-shape">
            
        </div>
        <div class="front card-shape">
            
        </div>
    </li>
    <li class="card clickable">
        <div class="back card-shape">
            
        </div>
        <div class="front card-shape">
            
        </div>
    </li>
    <li class="card clickable">
        <div class="back card-shape">
            
        </div>
        <div class="front card-shape">
            
        </div>
    </li>
    <li class="card clickable">
        <div class="back card-shape">
            
        </div>
        <div class="front card-shape">

```

```

        
    </div>
</li>
<li class="card clickable">
    <div class="back card-shape">
        
    </div>
    <div class="front card-shape">
        
    </div>
</li>
<li class="card clickable">
    <div class="back card-shape">
        
    </div>
    <div class="front card-shape">
        
    </div>
</li>
<li class="card clickable">
    <div class="back card-shape">
        
    </div>
    <div class="front card-shape">
        
    </div>
</li>
<li class="card clickable">
    <div class="back card-shape">
        
    </div>
    <div class="front card-shape">
        
    </div>
</li>
</ul>
<div class="message"></div>

```

Control Buttons and Options

- `info`: Contains control buttons for starting and resetting the game.
- `button`: The start button begins the game.
- `resetButton`: The reset button stops and resets the game.

- enableSound: A checkbox to enable or disable sound effects.
- enableLeaderboard: A checkbox to enable or disable the leaderboard.

```
<div class="info">
  <button>start</button>
  <button id="reset">Reset</button>
</div>
<label for="enableSound" style="color: antiquewhite; margin-bottom: 20px;">Sound</label>
<input type="checkbox" id="enableSound" checked>
<label for="enableLeaderboard" style="color: antiquewhite; margin-top: 20px;">Leaderboard</label>
<input type="checkbox" id="enableLeaderboard">
```

Leaderboard and Username Submission

- leaderboardContainer: Contains the leaderboard, which displays top scores.
- scroll, scrollObj: Elements for displaying the leaderboard content with scrollable functionality.
- leaderboard: The actual leaderboard where player scores are listed which is displayed if the checkbox is enabled.
- username-container: Contains an input field and a submit button for the player to enter their name to be recorded in the leaderboard.
- script: Links to the external JavaScript file (index.js) which contains the game logic.

```
<div id="leaderboardContainer" style="display: none;">
  <div class="scroll">
    <div class="scrollObj">
      <h2 style="font-weight: bold; margin-bottom: 20px;">Leaderboard</h2>
      <div id="leaderboard">
      </div>
    </div>
  </div>
</div>

<div class="username-container" style="display: none;">
  <input type="text" id="username" required>
  <div class="placeholder">Enter your name</div>
  <button id="submitUsername">Submit</button>
</div>
```



```
<script src="index.js"></script>
</body>
</html>
```

CSS CODE

This CSS provides comprehensive styling for a card matching game, including layout, animations, hover effects, input fields, and custom controls, all contributing to an engaging user experience.

- General Styles

Universal Selector (*):

This resets margin and padding for all elements to 0 and sets `box-sizing` to `border-box` for consistent sizing.

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
```

Body:

The body is centered both horizontally and vertically using flexbox. Elements are stacked vertically. The background color is a dark gray, and the font is set to 'Courier New'.

```
body {
  display: flex;
  align-items: center;
  justify-content: center;
  flex-direction: column; /*stack elements vertically */
  min-height: 100vh;
  background: #424245;
  font-family: 'Courier New', Courier, monospace;
}
```

Info Section

- **Info Container:**

This container is a flexbox with a gap between items, centered items, and even spacing. It has a fixed width of 550px.

```
.info {
```

```

display: flex;
gap: 20px;
align-items: center;
font-size: 22px;
justify-content: space-evenly;
width: 550px; /* info container the same width as the game container
*/
}

```

Buttons

- **General Button Styles:**

Buttons have an animation on load, are styled with a dark background, white text, rounded corners, padding, and no border. They use a specific font, have a margin, and feature a border and box-shadow transition on hover.

```

button {
  animation: slide 1s ease-in;
  position: relative;
  background: #282a3a;
  color: #fff;
  border-radius: 5px;
  padding: 10px 30px;
  border: 0;
  cursor: pointer;
  font-family: 'Courier New', Courier, monospace;
  font-size: 20px;
  font-weight: bold;
  margin-top: 20px ;
  margin-bottom: 20px;
  display: inline-block;
  border: 2px solid aquamarine;
  text-decoration: none;
  z-index: 1;
  overflow: hidden;
  transition: color 1s, box-shadow 1s;
  transition-delay: 0s, 1s;
}

```

- **Reset Button:**

Initially, the reset button is hidden.

```
#reset{
```

```
display: none;
}
```

Button Hover Effects:

On hover, buttons show an aquamarine shadow and an animated aquamarine background.

```
button:hover{
  transition-delay: 0s, 1s;
  color: #fff;
  box-shadow: 0 0 10px aquamarine,
              0 0 20px aquamarine,
              0 0 40px aquamarine,
              0 0 60px aquamarine,
              0 0 80px aquamarine,
              0 0 160px aquamarine;
}

button::before{
  content: '';
  position: absolute;
  top: 0;
  left: -50px;
  width: 0;
  height: 100%;
  background: aquamarine;
  transform: skewX(35deg);
  transition: 1s;
  z-index: -1;
}

button:hover:before{
  width: 100%;
}
```

Game State

- **State Container:**

Initially hidden, this container has a gap between items, bold text, a margin, and light gray text.

```
.state {
  animation: slide 0.7s ease-in;
  display: none;
  gap: 40px;
```

```
font-weight: bold;
margin-bottom: 25px;
color: #e6e6e6;
transition: 0.3s linear;
}
```

Game Container

- **General Game Container**

This container is initially hidden, has a fixed size, a light background, rounded corners, padding, and a transition effect.

```
.game-container {
  animation: slide 0.7s ease-in;
  display: none;
  height: 550px;
  width: 550px;
  background: #e6e6e6;
  border-radius: 10px;
  padding: 25px;
  transition: 0.3s linear;
}
```

Cards

- **All Cards Container and Card Styles:**

Cards and their containers are centered using flexbox. The cards have a perspective effect for 3D rotation, and transform styles for flipping animations.

This keyframe animation is used for shaking the card when not matched.

```
.all-cards, .card, .card-shape {
  display: flex;
  align-items: center;
  justify-content: center;
}

.all-cards {
  height: 100%;
  width: 100%;
  flex-wrap: wrap;
  justify-content: space-between;
```

```
}

.all-cards .card {
  cursor: pointer;
  position: relative;
  perspective: 800px;
  transform-style: preserve-3d;
  height: calc(100% / 4 - 10px);
  width: calc(100% / 4 - 10px);
}

.all-cards .card:hover{
  box-shadow: 0px 0px 5px 5px aquamarine;
}

.card {
  transition: transform 0.5s;
}

.card .back {
  background-color: rgb(7, 1, 22);
}

.card .front {
  transform: rotateY(-180deg);
  background-color: #d1d1e4;
}

.card.flipping .front {
  transform: rotateY(0deg);
}

.card.flipping .back {
  transform: rotateY(180deg);
}

.card.notMatchedShake {
  animation: shake 0.35s ease-in-out;
}

@keyframes shake {
  0%, 100% {
    transform: translateX(0);
  }
  20% {
    transform: translateX(-13px);
  }
}
```

```

    }
    40% {
        transform: translateX(13px);
    }
    60% {
        transform: translateX(-8px);
    }
    80% {
        transform: translateX(8px);
    }
}

```

Card Shape

- **Card Shape and Image Styles**

Card shapes and images are styled for consistent size, non-selectable, with a backface-visibility setting for 3D effect.

```

.all-cards .card .card-shape {
    width: 100%;
    height: 100%;
    user-select: none; /* To select all the li not only the image */
    pointer-events: none;
    position: absolute;
    border-radius: 7px;
    backface-visibility: hidden;
    transition: transform 0.35s linear;
    box-shadow: 2px 3px 10px rgba(39, 7, 92, 0.8);
}

.front-image {
    max-width: 65px;
    object-fit: cover;
}

.back-image {
    max-width: 120px;
    object-fit: cover;
}

```

Messages

- **Message Container**

Initially hidden, this container shows messages with a dark semi-transparent background, bold white text, and centered position.

```
.message {
  background-color: rgba(100,100,100,0.6);
  display: none;
  font-size: 40px;
  font-weight: bold;
  text-align: center;
  color: #fff;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  z-index: 100;
  position: fixed;
  justify-content: center;
  align-items: center;
  flex-direction: column;
}
```

Welcome Section:

The welcome section styles the introductory text with large, centered, colored text.

The keyframe for transition in of the buttons and board and images when the game starts.

```
.welcome{
  font-size: 2em;
  display: flex;
  flex-direction: column;
  align-items: center;
  margin-bottom: 50px;
}

h1{
  color: aquamarine;
  font-weight: bold;
}

h2{
  color: aquamarine;
  font-weight: normal;
  text-align: center;
}
```

```

}
h3{
  color: antiquewhite;
  margin-top: 70px;
  font-size: 0.7em;
  font-style: italic;
}
@keyframes slide {
  from{
    transform: translateX(-300px);
  }

  to{
    transform: translateX(0);
  }
}

```

Countdown

- **Countdown Timer**

This centers the countdown timer with large text and a semi-transparent background.

```

.countdown {
  position: fixed;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  font-size: 100px;
  color: aquamarine;
  font-weight: bold;
  text-align: center;
  z-index: 200;
  background-color: rgba(100,100,100,0.6);
}

```

Leaderboard

- **Leaderboard Container**

This styles the leaderboard with a scrollable container and specific list item styles.


```

#leaderboardContainer {
    margin-top: 20px;
    background: #282a3a;
    width: 700px;
    padding: 30px;
    border-radius: 50px;
}

.scroll{
    width: 650px;
    background: #282a3a;
    height: 300px;
    overflow: hidden;
    overflow-y: scroll ;
}

.scrollObj{
    color: antiquewhite;
    padding: 20px;
}

#leaderboard {
    list-style: none;
    padding: 0;
}

#leaderboard li {
    font-size: 18px;
    padding: 5px 0;
}

#enableLeaderboard{
    margin-top: 20px;
    margin-bottom: 20px;
}

```

Username Input

- Username Container

This section styles the input field and placeholder for entering a username which is hidden at first then appear when the game end.

```
.username-container {
  top: 150px;
  left: 0;
  right: 0;
  bottom: 0;
  z-index: 100;
  position: fixed;
  justify-content: center;
  align-items: center;
  flex-direction: column;
  display: none;
}

#submitUsername{
  margin-top: 150px;
}

.username-container input[type="text"] {
  position: absolute;
  padding: 10px;
  font-size: 16px;
  border-radius: 10px;
  border: 2px solid #ccc;
  margin-right: 10px;
  outline: none;
  background: transparent;
  transition: 0.1s ease;
  z-index: 1111;
  line-height: 20px;
  width: 300px;
}

.placeholder{
  position: absolute;
  font-size: 20px;
  color: #eee;
  padding: 0 10px;
  margin: 10px 10px;
  transition: 0.2s ease;
}

.username-container input[type="text"]:focus + .placeholder
, .username-container input[type="text"]:valid + .placeholder{
```

```

    color: aquamarine;
    height: 30px;
    line-height: 30px;
    transform: translate(-55px,-30px) scale(0.88);
    z-index: 1111;
    font-weight: bold;
    background-color: rgba(100,100,100,0);
}

.username-container input[type="text"]:focus,
.username-container input[type="text"]:valid{
    color: aquamarine;
    border: 4px solid aquamarine;
}

.username-container button {
    padding: 10px 20px;
    font-size: 16px;
    background-color: #282a3a;
    color: #fff;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s ease;
}

.username-container button:hover {
    background-color: #1c1e2a;
}

```

Checkbox

- **Checkbox Styles**

This section styles the custom checkbox, including transitions for its checked and unchecked states.

```

input[type="checkbox"]{
    position: relative;
    width: 120px;
    height: 40px;
    -webkit-appearance: none;
    background: linear-gradient(0deg , #333,#000);
    outline: none;
    border-radius: 20px;
}

```

```
    box-shadow: 0 0 0 4px #353535, 0 0 0 5px #3e3e3e, inset 0 0 10px
    rgba(0,0,0,1), 0 5px 20px rgba(0,0,0,0.5), inset 0 0 15px rgba(0,0,0,0.2);
}
```

```
input:checked[type="checkbox"]{
    background: linear-gradient(0deg , rgb(121, 230, 193),aquamarine);
    box-shadow: 0 0 2px rgb(121, 230, 193),0 0 0 4px #353535, 0 0 0 5px
    #3e3e3e, inset 0 0 10px rgba(0,0,0,1), 0 5px 20px rgba(0,0,0,0.5), inset 0
    0 15px rgba(0,0,0,0.2);
}
```

```
input[type="checkbox"]:before{
    content: '';
    position: absolute;
    top: 0;
    left: 0;
    width: 80px;
    height: 40px;
    background: linear-gradient(0deg , #000,#6b6b6b);
    border-radius: 20px;
    box-shadow: 0 0 0 1px #232323;
    transform: scale(0.98,0.97);
    transition: 0.5s;
}
```

```
input:checked[type="checkbox"]:before{
    left: 40px;
}
```

```
input[type="checkbox"]:after{
    content: '';
    position: absolute;
    top: 18px;
    left: 65px;
    width: 4px;
    height: 4px;
    background: linear-gradient(0deg, #6b6b6b,#000);
    border-radius: 50%;
    transition: 0.5s;
}
```

```
input:checked[type="checkbox"]:after{
```

```
background: rgb(88, 172, 144);  
left: 105px;  
box-shadow: 0 0 5px aquamarine, 0 0 15px aquamarine ;  
}
```

Points

- **Points Display**

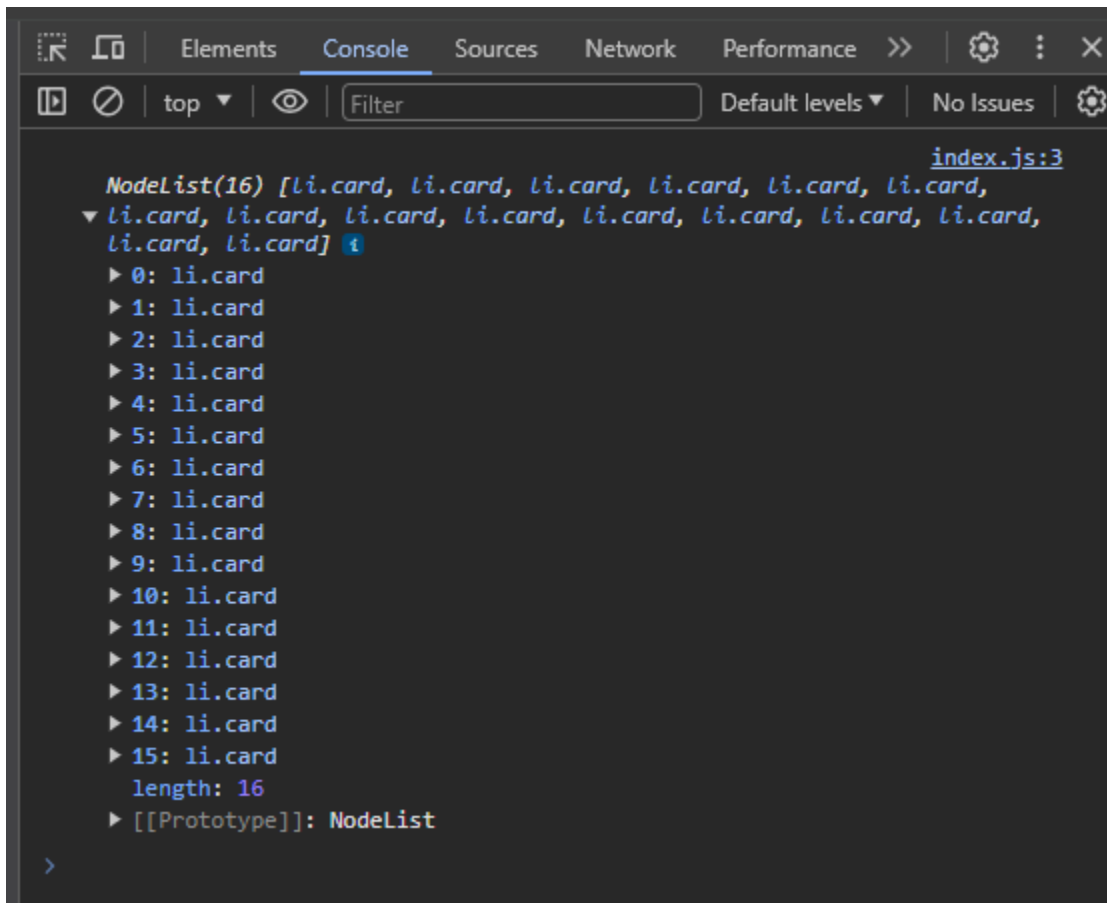
This section styles the points display list.

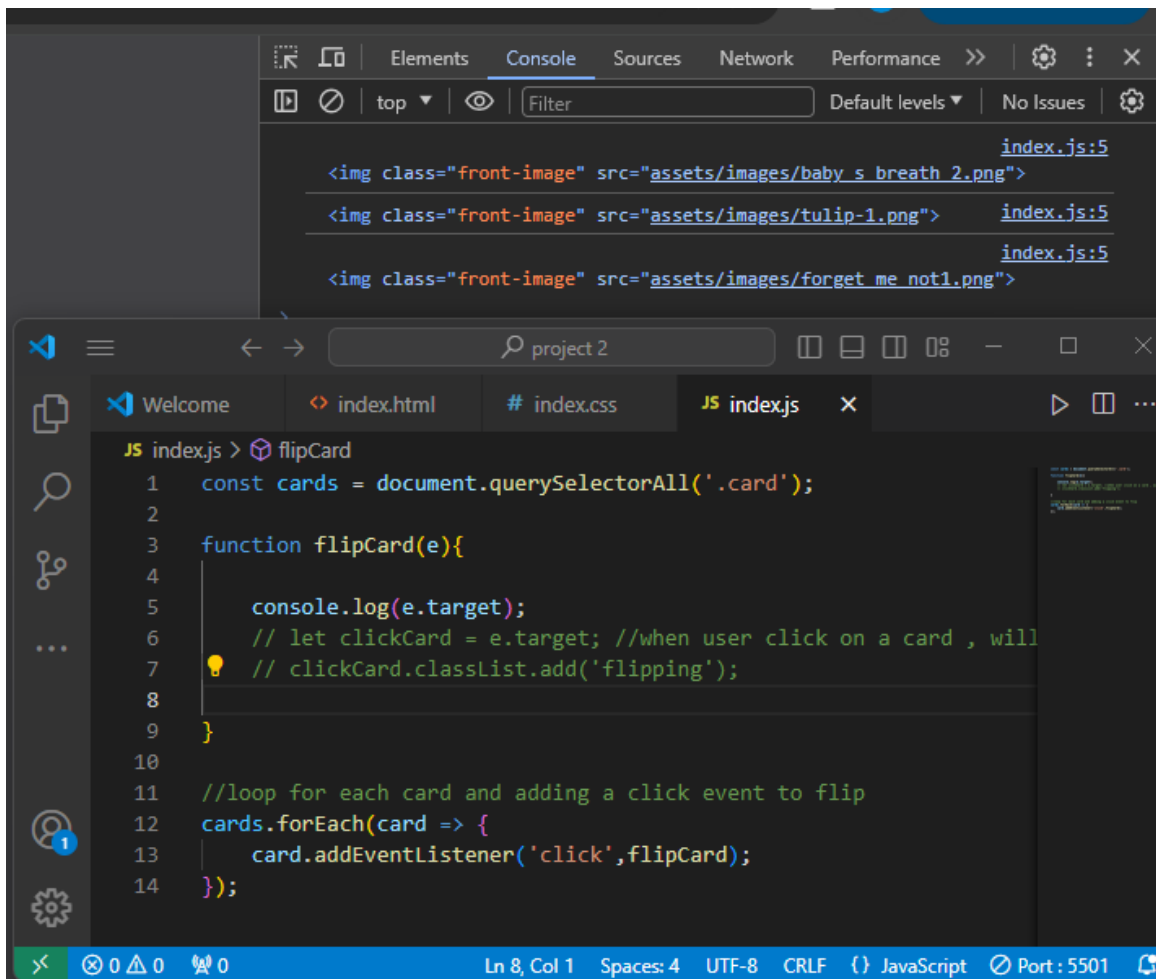
```
.points{  
  display: block;  
  font-size: 0.5em;  
  color: antiquewhite;  
  margin-top: 30px;  
}  
  
.points li{  
  list-style-type: none;  
  margin-top: 15px;  
}
```

JAVASCRIPT CODE

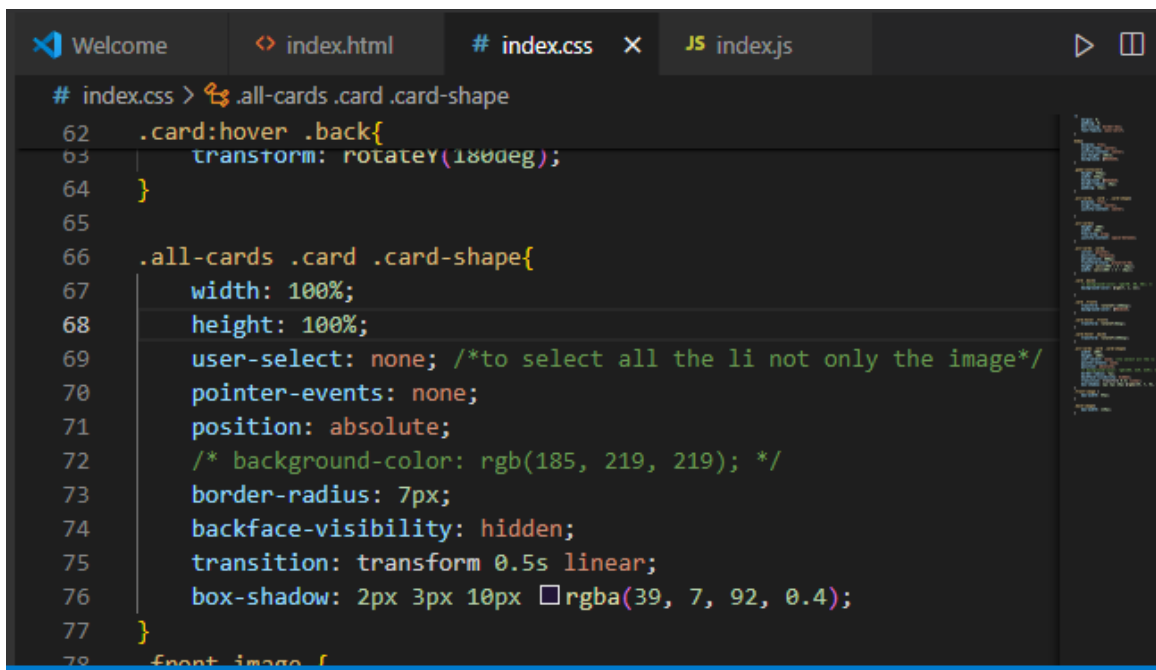
This is how I begin to construct the code to be able to build the skeleton base then add some features and functions.

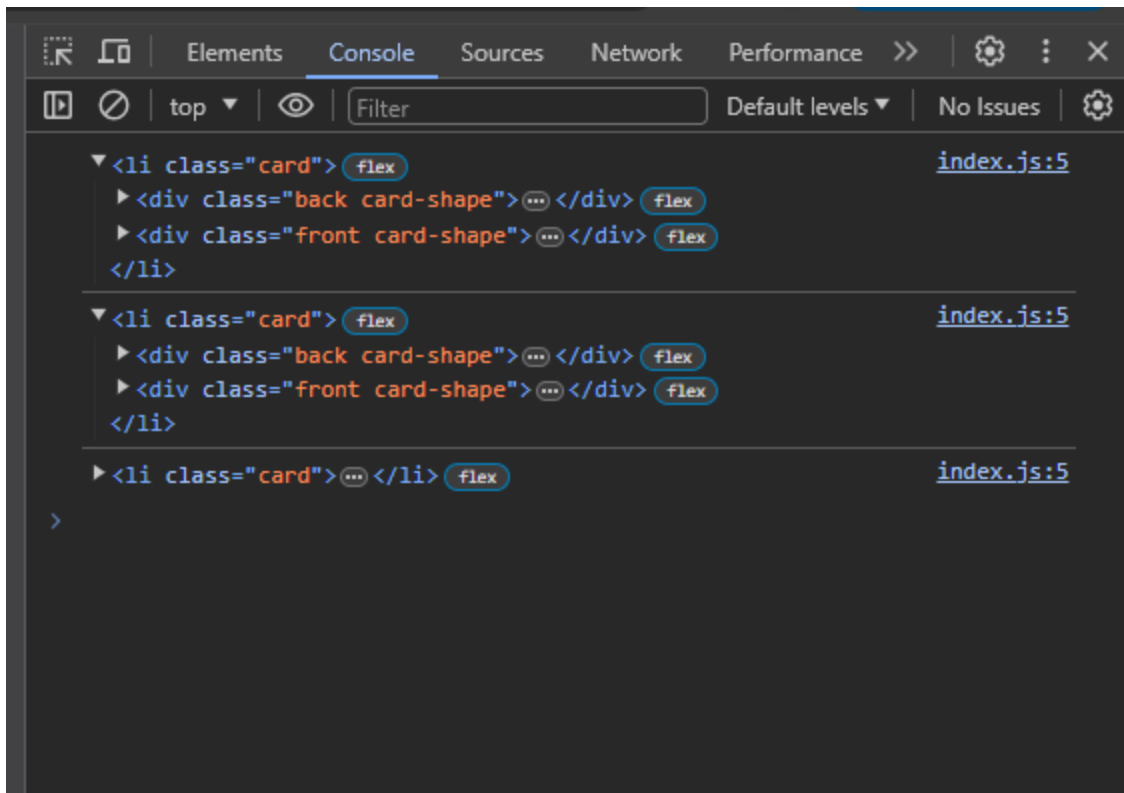
```
JS index.js > ...
1  const cards = document.querySelectorAll('.card');
2
3  console.log(cards)
4
```



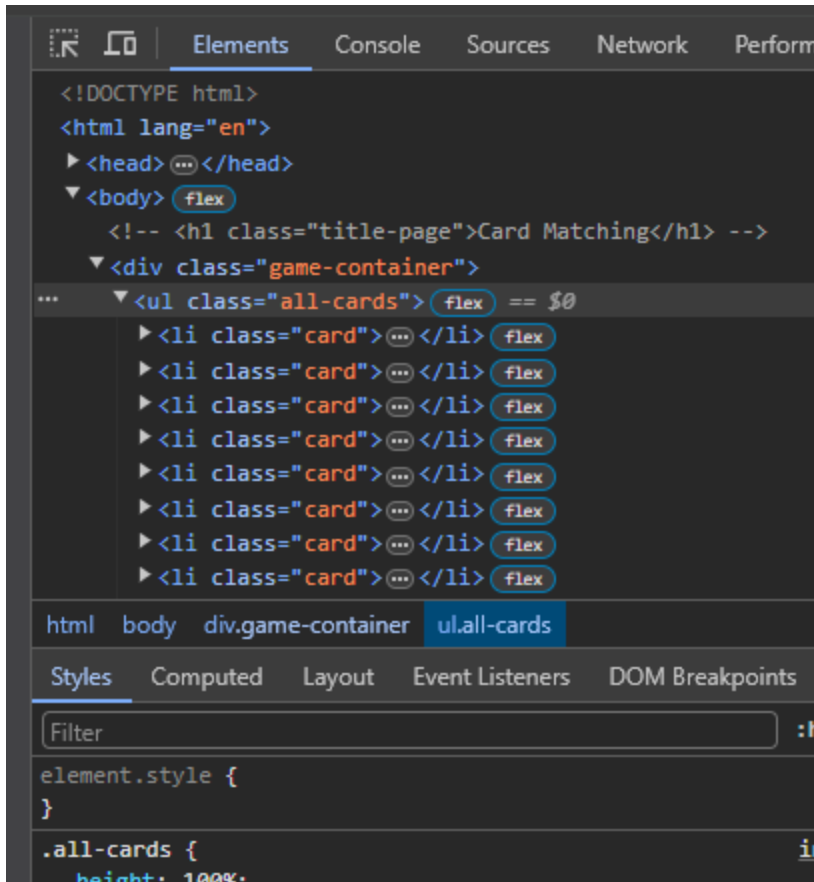


Want to select the li not only the image

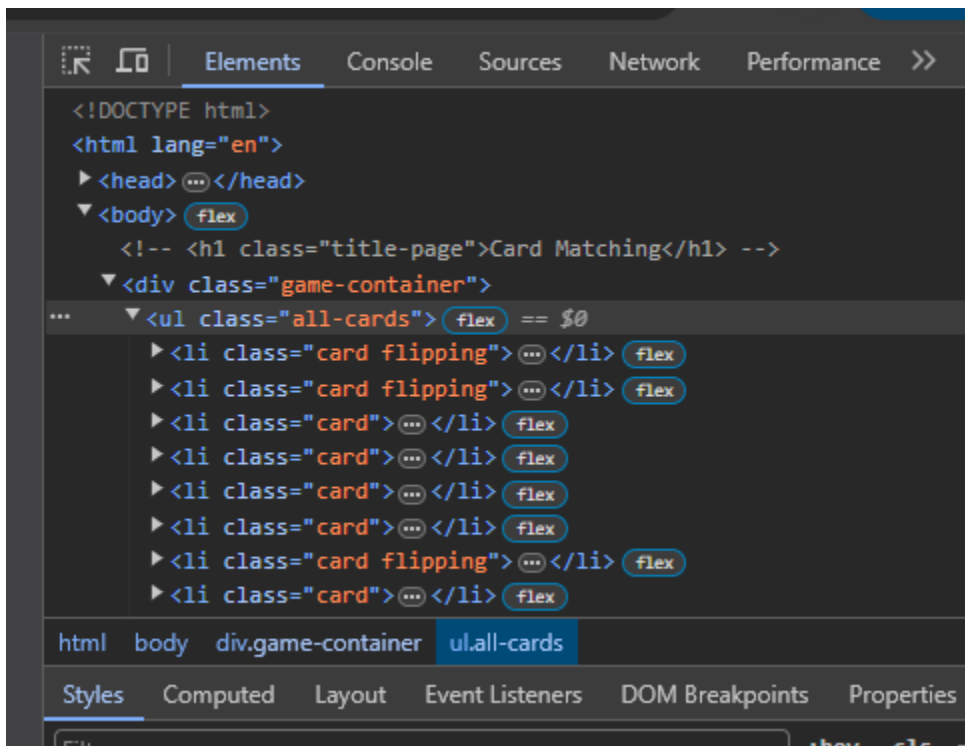




```
JS index.js > ...
1  const cards = document.querySelectorAll('.card');
2
3  function flipCard(e){
4
5      let clickCard = e.target; //when user click on a card , will be added to flipping class
6      clickCard.classList.add('flipping');
7
8  }
9
10 //loop for each card and adding a click event to flip
11 cards.forEach(card => {
12     card.addEventListener('click',flipCard);
13 });
```

Clicked to add it in flipping class



```

52
53  ✓ .card .front{
54      transform: rotateY(-180deg);
55      background-color: #d1d1e4;
56  }
57
58  ✓ .card.flipping .front{
59      transform: rotateY(0deg);
60  }
61
62  ✓ .card.flipping .back{
63      transform: rotateY(180deg);
64  }

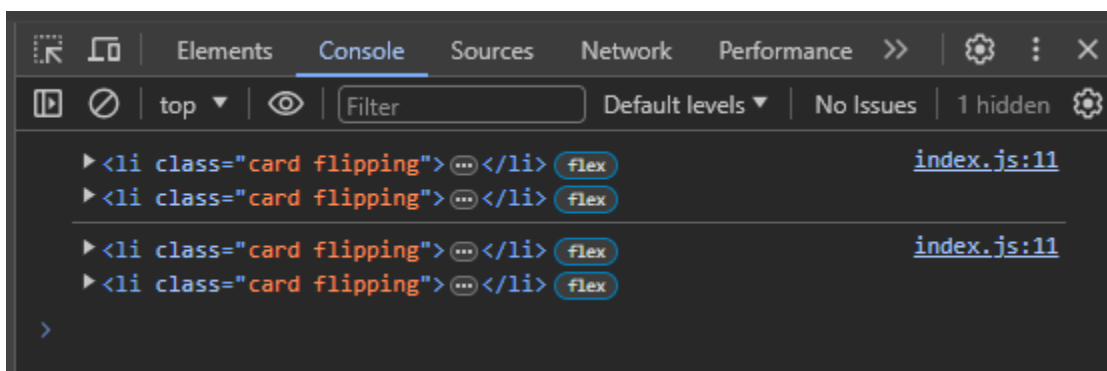
```

```

JS index.js > flipCard
1  const cards = document.querySelectorAll('.card');
2
3  let firstCard , secondCard;
4
5  function flipCard(e){
6
7      let clickCard = e.target; //when user click on a card , will
8      clickCard.classList.add('flipping');
9      firstCard = clickCard;
10     secondCard = clickCard;
11     console.log(firstCard,secondCard)
12
13 }
14

```

Trying 2 match 2 cards By clicking 2 images getting 4 cards



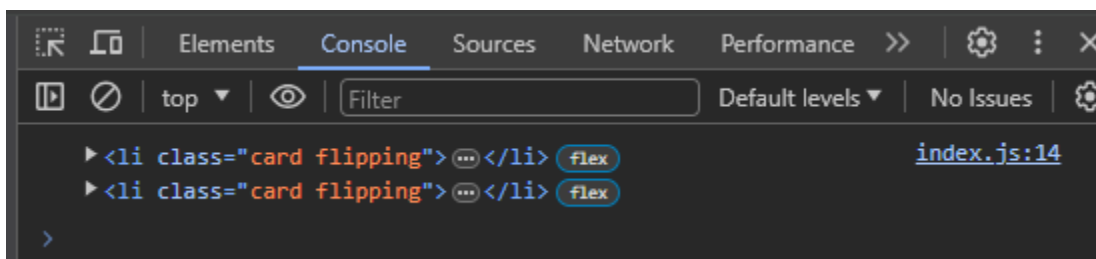
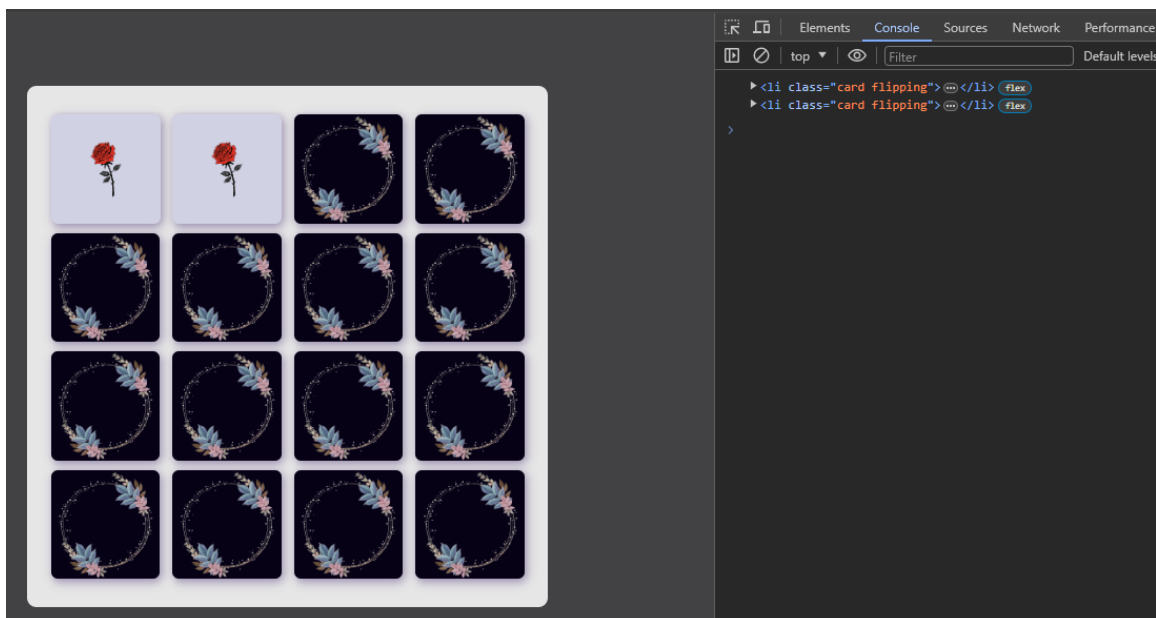
```

▶ <li class="card flipping">...</li> flex index.js:11
▶ <li class="card flipping">...</li> flex index.js:11
▶ <li class="card flipping">...</li> flex index.js:11
▶ <li class="card flipping">...</li> flex index.js:11
>

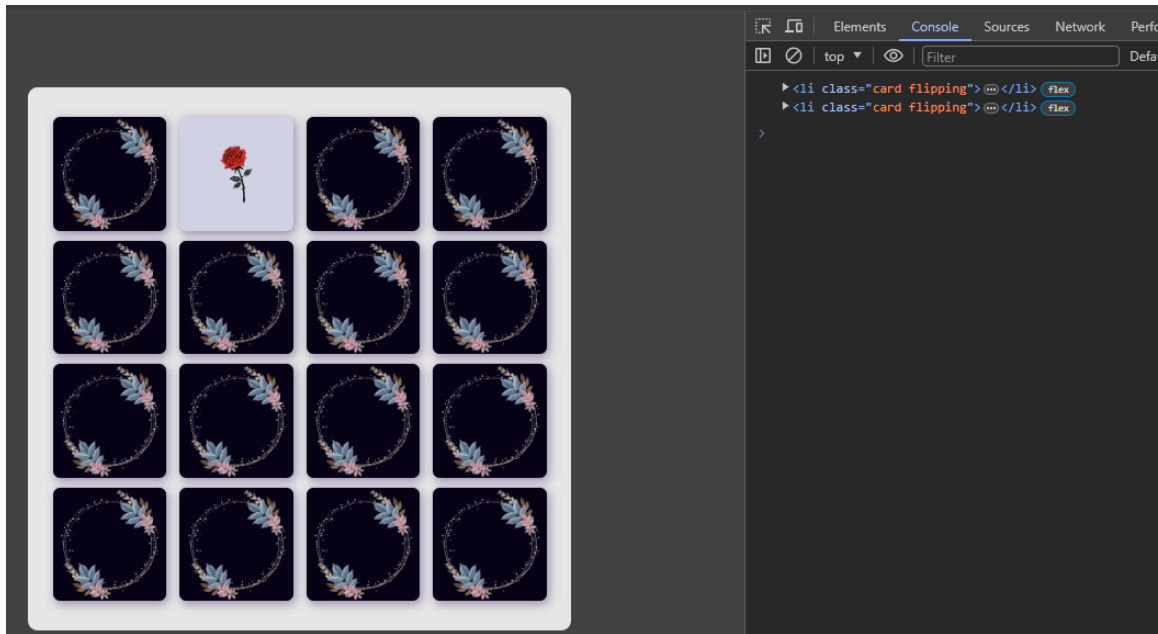
```

return first card value when clicked 2 cards

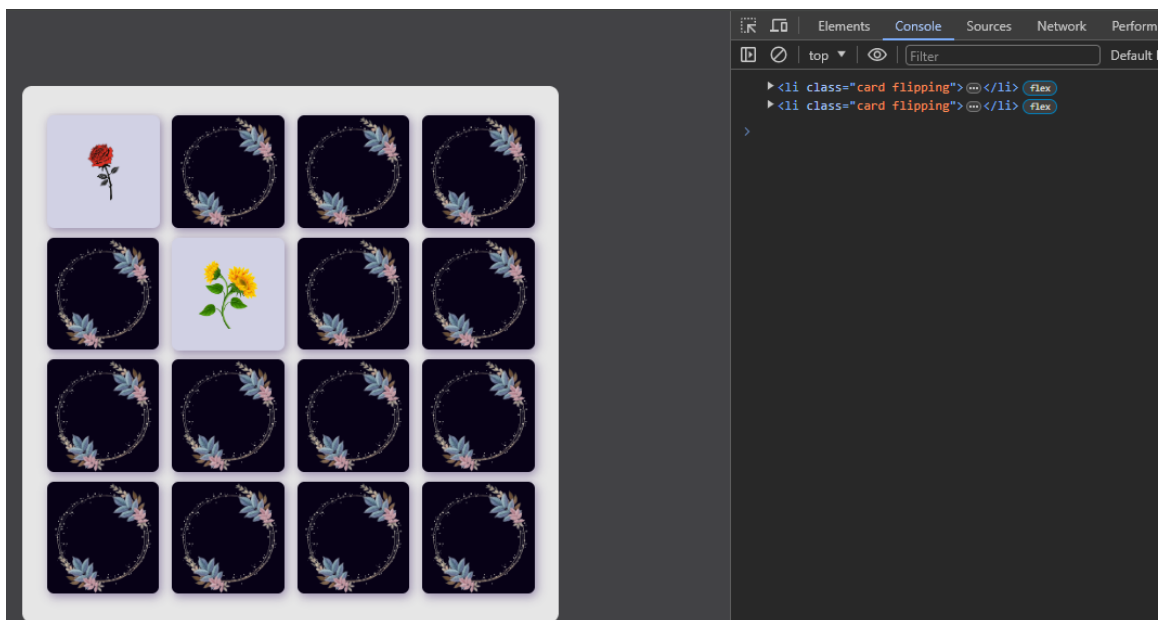
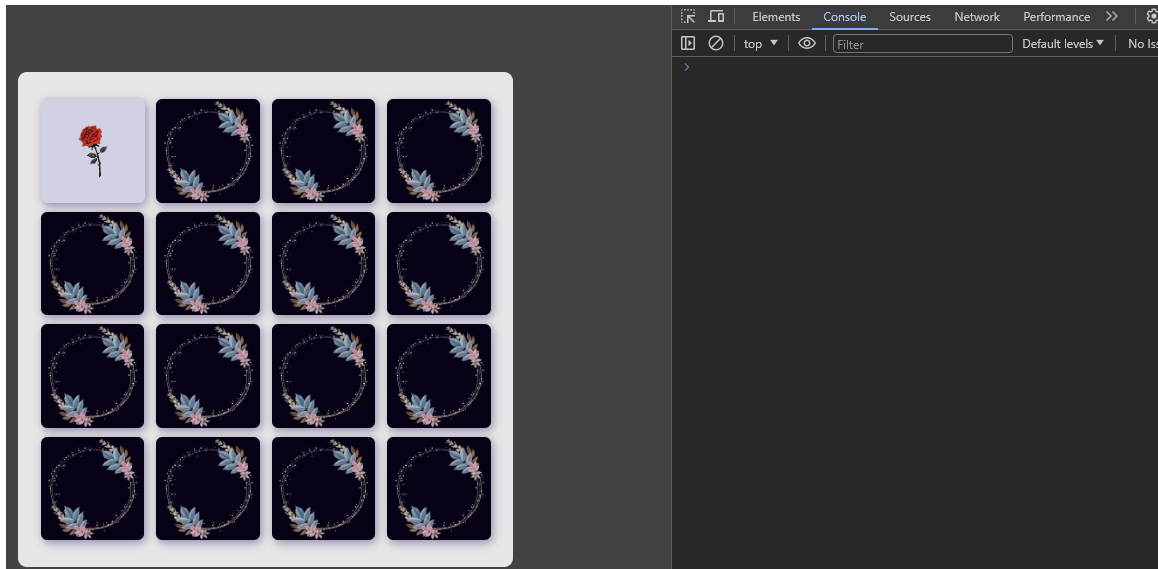
```
JS index.js > ...
3 let firstCard , secondCard;
4
5 function flipCard(e){
6
7     let clickCard = e.target; //when user click on a card , will be
8     clickCard.classList.add('flipping');
9
10    //return first card value when clicked
11    if(!firstCard){
12        return firstCard = clickCard;
13    }
14    secondCard = clickCard;
15    console.log(firstCard,secondCard)
16    // where the image
```



Prevent from clicking on one card 2 times

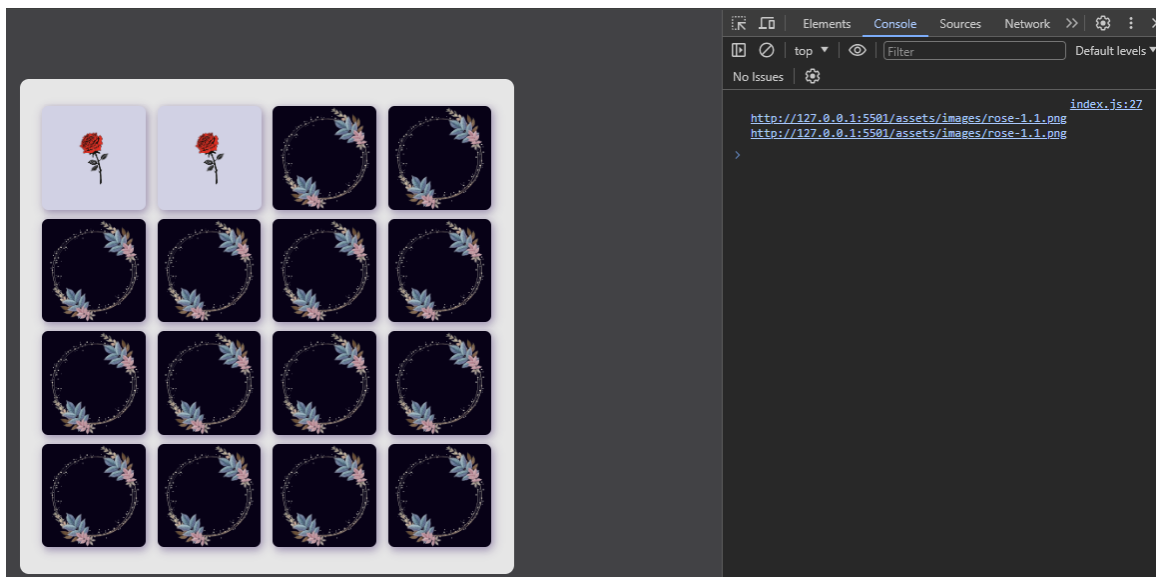


```
JS index.js > flipCard
1  const cards = document.querySelectorAll('.card');
2
3  let firstCard , secondCard;
4
5  function flipCard(e){
6      let clickCard = e.target; //when user click on a card , will be
7      if(clickCard !== firstCard){
8          clickCard.classList.add('flipping');
9          //return first card value when clicked
10         if(!firstCard){
11             return firstCard = clickCard;
12         }
13         secondCard = clickCard;
14         console.log(firstCard,secondCard)
15         // where the image
16         // let firstCardImage = firstCard.querySelector('img'),
17         // secondCardImage = secondCard.querySelector('img');
18         // matchingCards(firstCardImage,secondCardImage);
19     }
20 }
21 }
```



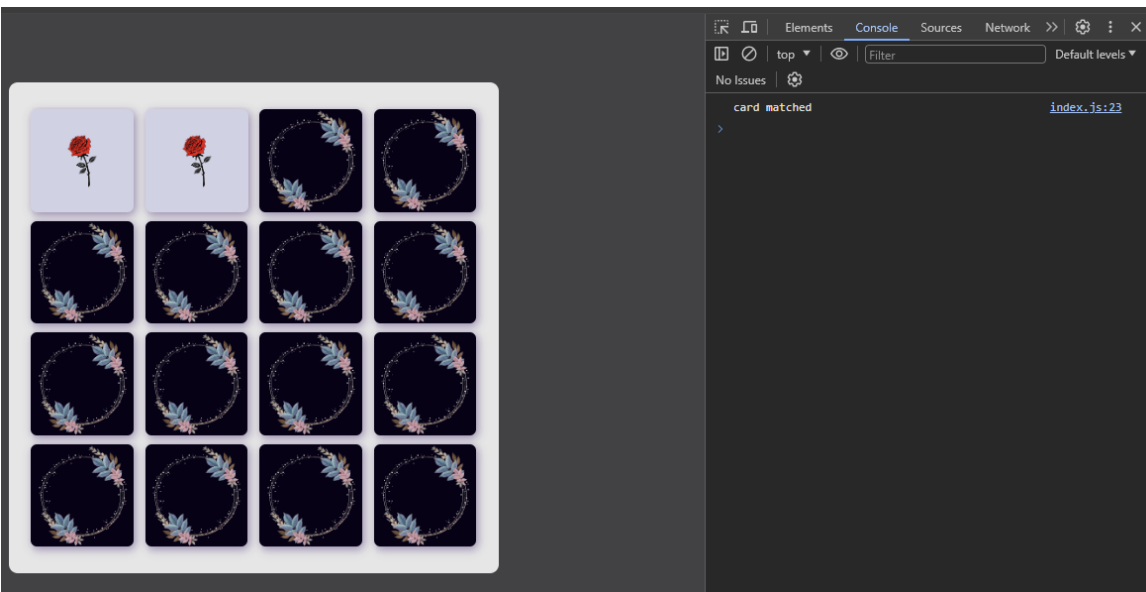
Getting the images to match it

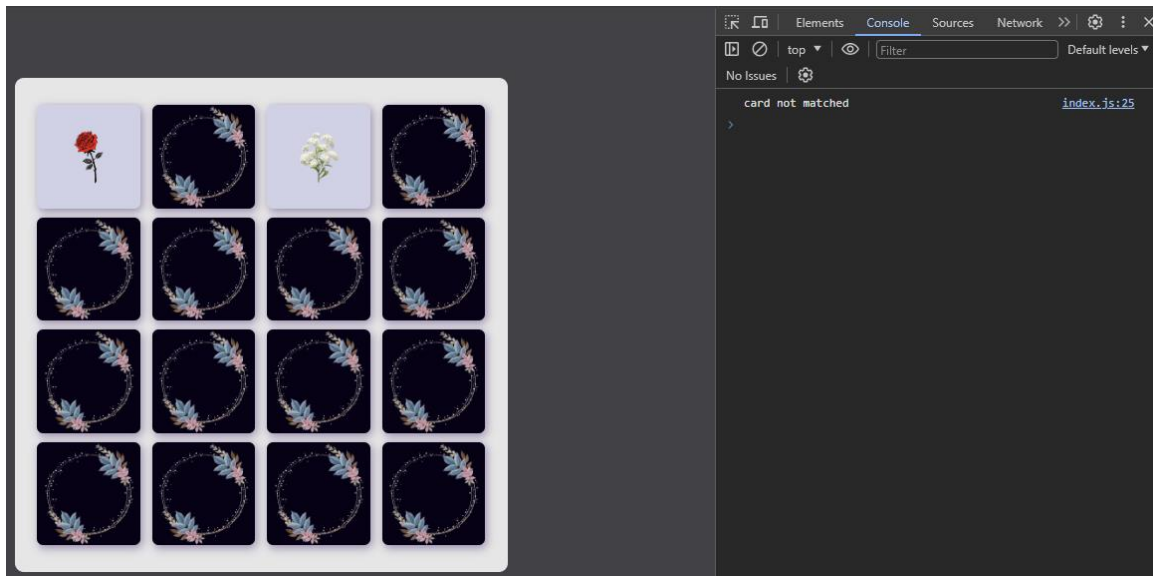
```
JS index.js > matchingCards
1  const cards = document.querySelectorAll('.card');
2
3  let firstCard , secondCard;
4
5  function flipCard(e){
6      let clickCard = e.target; //when user click on a card , will be added to flip
7      if(clickCard !== firstCard){
8          clickCard.classList.add('flipping');
9          //return first card value when clicked
10         if(!firstCard){
11             return firstCard = clickCard;
12         }
13         secondCard = clickCard;
14         // where the image
15         let firstCardImage = firstCard.querySelector('.front img').src,
16         secondCardImage = secondCard.querySelector('.front img').src;
17         matchingCards(firstCardImage,secondCardImage);
18     }
19 }
20
21 function matchingCards(image1,image2){
22     // if(image1 === image2){
23     //     return console.log('card matched');
24     // }
25     // console.log('card')
26     console.log(image1,image2);
27 }
```



JS index.js > matchingCards

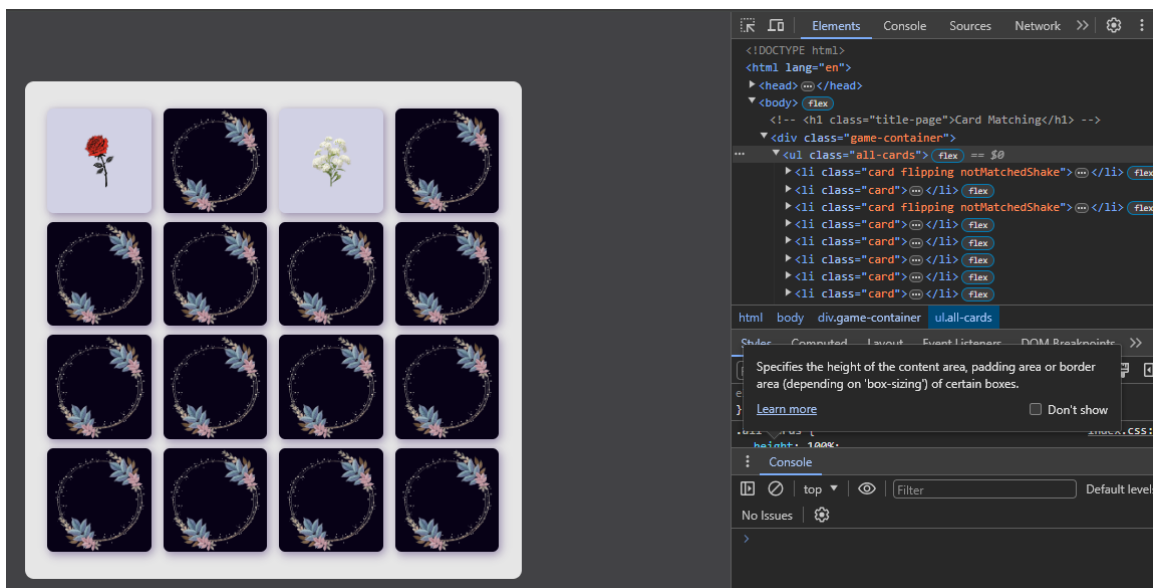
```
1  const cards = document.querySelectorAll('.card');
2
3  let firstCard , secondCard;
4
5  function flipCard(e){
6      let clickCard = e.target; //when user click on a card , will be added to flip
7      if(clickCard !== firstCard){
8          clickCard.classList.add('flipping');
9          //return first card value when clicked
10         if(!firstCard){
11             return firstCard = clickCard;
12         }
13         secondCard = clickCard;
14         // where the image
15         let firstCardImage = firstCard.querySelector('.front img').src,
16         secondCardImage = secondCard.querySelector('.front img').src;
17         matchingCards(firstCardImage,secondCardImage);
18     }
19 }
20
21 function matchingCards(image1,image2){
22     if(image1 === image2){
23         return console.log('card matched');
24     }
25     console.log('card not matched');
26     // console.log(image1,image2);
27 }
28
```

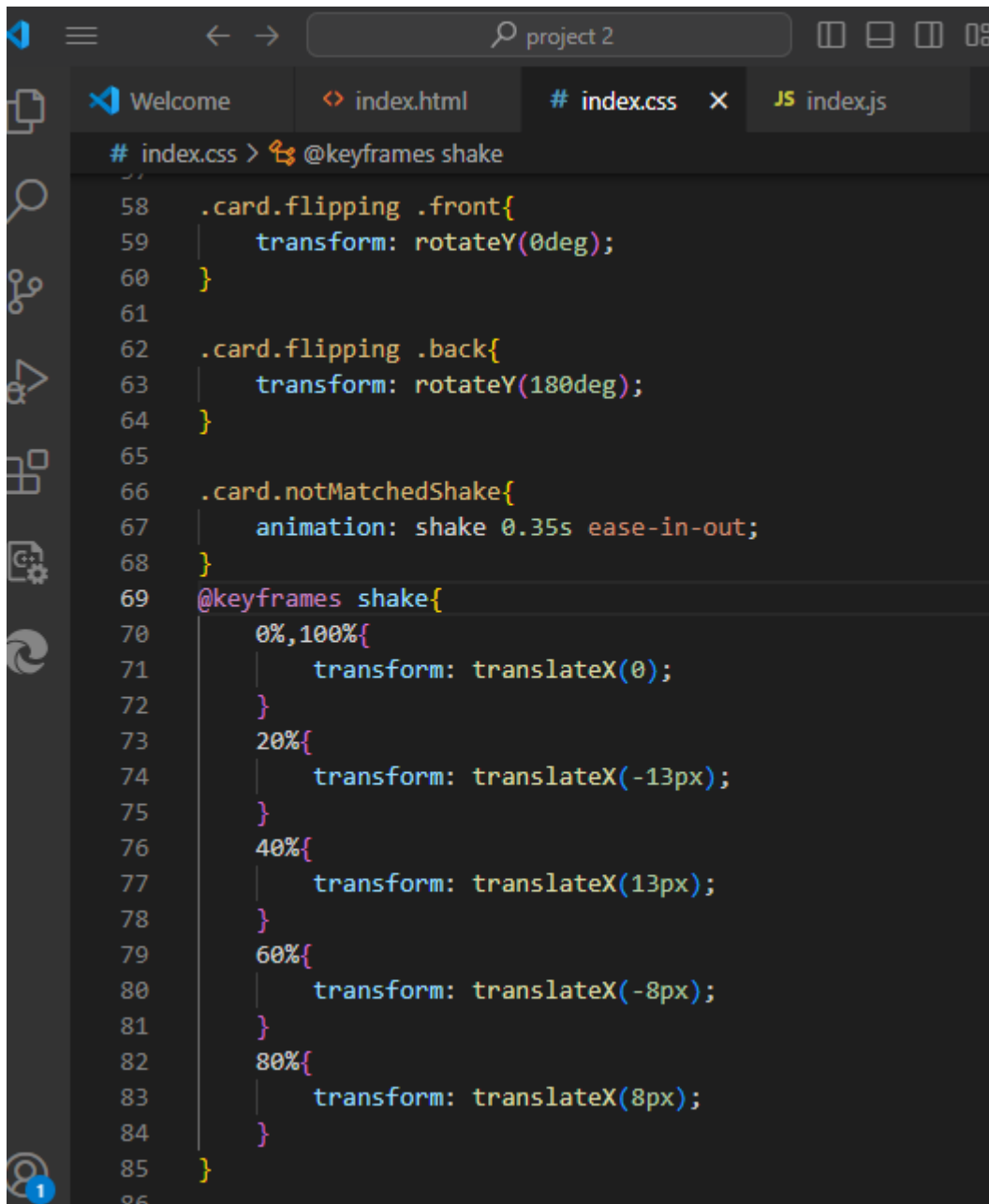




Added a class for Effect that 2 cards don't match

```
function matchingCards(image1,image2){
  if(image1 === image2){
    return console.log('card matched');
  }
  // console.log('card not matched');
  // console.log(image1,image2);
  firstCard.classList.add('notMatchedShake');
  secondCard.classList.add('notMatchedShake');
}
```





The image shows a web browser window with the address bar displaying "project 2". Below the browser, a code editor is open with three tabs: "Welcome", "index.html", and "# index.css" (which is the active tab). The code in the editor is CSS for a card flip animation. It includes rules for the front and back of the card, a shake animation, and a keyframe definition for the shake. The code is as follows:

```
# index.css > @keyframes shake
58 .card.flipping .front{
59     transform: rotateY(0deg);
60 }
61
62 .card.flipping .back{
63     transform: rotateY(180deg);
64 }
65
66 .card.notMatchedShake{
67     animation: shake 0.35s ease-in-out;
68 }
69 @keyframes shake{
70     0%,100%{
71         transform: translateX(0);
72     }
73     20%{
74         transform: translateX(-13px);
75     }
76     40%{
77         transform: translateX(13px);
78     }
79     60%{
80         transform: translateX(-8px);
81     }
82     80%{
83         transform: translateX(8px);
84     }
85 }
86
```

```

function matchingCards(image1,image2){
  if(image1 === image2){
    return console.log('card matched');
  }
  // console.log('card not matched');
  // console.log(image1,image2);
  //shake after 400ms if 2 cards doesnt match
  setTimeout(()=>{
    firstCard.classList.add('notMatchedShake');
    secondCard.classList.add('notMatchedShake');
  },400);
}

```

If not matched we will Remove from classes and setting to blank to be able to click in another 2 cards

```

20
21 function matchingCards(image1,image2){
22   if(image1 === image2){
23     return console.log('card matched');
24   }
25   // console.log('card not matched');
26   // console.log(image1,image2);
27   //shake after 400ms if 2 cards doesnt match
28   setTimeout(()=>{
29     firstCard.classList.add('notMatchedShake');
30     secondCard.classList.add('notMatchedShake');
31   },400);
32
33   //remove notmatchedshake and flipping classes from cards as not matched , after
34   setTimeout(()=>{
35     firstCard.classList.remove('notMatchedShake','flipping');
36     secondCard.classList.remove('notMatchedShake','flipping');
37     firstCard = secondCard = ''; //setting cards to blank to click on another 2 cards
38   },1200);
39
40 }
41

```

2 cards matched remove event listner from them so to dont click on them again

```

function matchingCards(image1,image2){
  if(image1 === image2){
    // return console.log('card matched');
    //2 cards matched remove event listener from them so to dont click on them again
    firstCard.removeEventListener('click',flipCard);
    secondCard.removeEventListener('click',flipCard);
    firstCard = secondCard = ''; //setting cards to blank to click on another 2 cards
    return;
  }
  // console.log('card not matched');
  // console.log(image1,image2);
  //shake after 400ms if 2 cards doesnt match
  setTimeout(()=>{
    firstCard.classList.add('notMatchedShake');
    secondCard.classList.add('notMatchedShake');
  },400);

  //remove notmatchedshake and flipping classes from cards as not matched , after
  setTimeout(()=>{
    firstCard.classList.remove('notMatchedShake','flipping');
    secondCard.classList.remove('notMatchedShake','flipping');
    firstCard = secondCard = ''; //setting cards to blank to click on another 2 cards
  },1200);
}

```

disable cards from multibleclick until cards are unflipped or matched

if match card is 8 , user matched all cards

after matching all cards ,will wait 1s then will flip all cards

```

Welcome  index.html  # index.css  JS index.js  X
JS index.js > matchingCards
1  let cards = Array.from(document.querySelectorAll('.card'));
2
3  let matchCard = 0;
4  let firstCard , secondCard;
5  //disable cards from multibleclick until cards are unflipped or matched
6  let disableMultibleClick = false;
7
8  function flipCard(e){
9    let clickCard = e.target; //when user click on a card , will be added to flipping class
10   if(clickCard !== firstCard && !disableMultibleClick){
11     clickCard.classList.add('flipping');
12     //return first card value when clicked
13     if(!firstCard){
14       return firstCard = clickCard;
15     }
16     secondCard = clickCard;
17     disableMultibleClick = true;
18     // where the image
19     let firstCardImage = firstCard.querySelector('.front img').src,
20     secondCardImage = secondCard.querySelector('.front img').src;
21     matchingCards(firstCardImage,secondCardImage);
22   }
23 }
24

```

```
JS index.js > ...
25 function matchingCards(image1,image2){
26     if(image1 === image2){
27         matchCard++;
28         //if match card is 8 , user matched all cards
29         if(matchCard == 8){
30             //after matching all cards ,will wait 1s then will flip all cards
31             setTimeout(()=>{
32                 return shuffleCard();
33             },1000);
34         }
35         // return console.log('card matched');
36         //2 cards matched remove event listner from them so to dont click on them again
37         firstCard.removeEventListener('click',flipCard);
38         secondCard.removeEventListener('click',flipCard);
39         firstCard = secondCard = ''; //setting cards to blank to click on another 2 cards
40         return disableMultibleClick = false;
41     }
42     // console.log('card not matched');
43     // console.log(image1,image2);
44     //shake after 400ms if 2 cards doesnt match
45     setTimeout(()=>{
46         firstCard.classList.add('notMatchedShake');
47         secondCard.classList.add('notMatchedShake');
48     },400);
49
50     //remove notmatchedshake and flipping classes from cards as not matched , after
51     setTimeout(()=>{
52         firstCard.classList.remove('notMatchedShake','flipping');
53         secondCard.classList.remove('notMatchedShake','flipping');
54         firstCard = secondCard = ''; //setting cards to blank to click on another 2 cards
55         disableMultibleClick = false;
56     },1200);
}
```

This is the explanation of final code

HTML Elements and Game Variables

1. Selecting HTML Elements:

- cards: Collects all elements with the class card. These represent the memory cards in the game.
- timeCountDown, flipsCount, button, resetButton, message: Elements used to display the timer, flip count, control buttons, and game messages.
- clickSound, matchSound, victorySound, gameOverSound: Audio elements that play sounds during different game events.
- scoreDisplay: Displays the current score.
- enableLeaderboardCheckbox, leaderboardElement, leaderboardContainer: Elements related to enabling/disabling and displaying the leaderboard.
- usernameContainer, usernameInput, submitUsernameButton: Elements for submitting the player's username to the leaderboard.
- isWaitingForUsername: A boolean flag indicating whether the game is waiting for the player to submit their username.

2. Game State Variables:

- matchCard: Tracks the number of matched pairs.

- firstCard, secondCard: Hold references to the currently selected cards.
- time, startTime, maxTime, timeLeft: Variables related to the game's timer.
- flips: Counts the number of card flips made by the player.
- isPlaying: Boolean indicating if the game is currently active.
- disableMultipleClick: Boolean to prevent multiple cards from being clicked simultaneously.
- countdownActive: Boolean to indicate if the countdown is active.
- score: Tracks the player's score.
- consecutiveMatches: Tracks consecutive matches to implement bonus scoring.

```
let cards = Array.from(document.querySelectorAll('.card'));
const timeCountDown = document.querySelector('.timer'),
    flipsCount = document.querySelector('.flips'),
    button = document.querySelector('button'),
    resetButton = document.getElementById('reset'),
    message = document.querySelector('.message'),
    clickSound = document.getElementById('clickSound'),
    matchSound = document.getElementById('matchSound'),
    victorySound = document.getElementById('VictorySound'),
    countdownButton = document.getElementById('countdownBtn'),
    gameOverSound = document.getElementById('gameOverSound'),
    scoreDisplay = document.querySelector('.score'),
    enableLeaderboardCheckbox =
document.getElementById('enableLeaderboard'),
    leaderboardElement = document.getElementById('leaderboard'),
    leaderboardContainer = document.getElementById('leaderboard'); //
Container to display the leaderboard

const usernameContainer = document.querySelector('.username-container');
const usernameInput = document.getElementById('username');
const submitUsernameButton = document.getElementById('submitUsername');

let isWaitingForUsername = false; // Add a flag to track username
submission

let matchCard = 0;
let firstCard, secondCard, time, startTime;
let maxTime = 60;
let timeLeft = maxTime;
let flips = 0;
let isPlaying = false;
```

```
let disableMultipleClick = true;
let countdownActive = false; // variable to track the countdown state
let score = 0;
let consecutiveMatches = 0;
```

Game Initialization Functions

3. Reset Game:

- `resetGame()`: This function resets the game state, including the timer, flips, score, and card visibility. It shuffles the cards to ensure a random order for the new game.

```
function resetGame() {
  document.querySelector('.game-container').style.display='block';
  document.querySelector('.state').style.display='flex';
  document.querySelector('#reset').style.display='block';
  document.querySelector('.welcome').style.display='none';

  usernameInput.value = '';
  usernameContainer.style.display = 'none'; // Hide the username input
  field

  flips = matchCard = score = consecutiveMatches = 0;
  timeLeft = maxTime;
  firstCard = secondCard = '';
  clearInterval(time); //stop timer
  timeCountDown.innerText = `Timer: ${timeLeft}`;
  flipsCount.innerText = `Flips count: ${flips}`;
  scoreDisplay.innerText = `Score: ${score}`;
  isPlaying = false;
  disableMultipleClick = true;

  // Reset card classes and visibility
  cards.forEach((card) => {
    card.classList.remove('matched', 'flipping', 'notMatchedShake');
    card.style.visibility = 'visible';
    card.addEventListener('click', flipCard);
  });

  // Shuffle cards
  shuffleCard();
}
```

Shuffle Cards:

- `shuffleCard()`: Randomly shuffles the cards by changing their order

```
function shuffleCard() {
  let indices = Array.from(Array(cards.length).keys());
  indices.sort(() => Math.random() - 0.5);

  cards.forEach((card, index) => {
    card.style.order = indices[index];
  });
}
```

Display Message:

`displayMessage(msg)`: Displays a message and updates the control buttons based on the game state.

```
function displayMessage(msg) {
  if (!isWaitingForUsername) {
    message.innerText = msg;
    message.style.display = 'flex';
    button.innerText = 'start'; // Change button text to "start" when
    game is over or victory is achieved
    resetButton.disabled = true; // Disable reset button when
    displaying message
  }
}
```

Timer Initialization:

- `initTimer()`: Decreases the timer every second and checks if the time has run out to end the game.

```
function initTimer() {
  if (timeLeft <= 0) {
    clearInterval(time);
    playGameOverSound();
    endGame('Game Over', flips, maxTime - timeLeft, score);
    return;
  }
}
```

```

    timeLeft--;
    timeCountDown.innerText = `Timer: ${timeLeft}`;
}

```

Play Sound Effects:

- Functions to play specific sound effects when cards are clicked, matched, or when the game ends.

```

function playClickSound() {
    clickSound.currentTime = 0;
    clickSound.play();
}

function playMatchSound() {
    matchSound.currentTime = 0;
    matchSound.play();
}

function playVictorySound() {
    victorySound.currentTime = 0;
    victorySound.play();
}

function playGameOverSound() {
    gameOverSound.currentTime = 0;
    gameOverSound.play();
}

```

Card Interaction

Flipping Cards:

- `flipCard({ target: clickCard })`: Handles the logic for flipping cards, ensuring that the game rules are followed, and updates the flip count.

```

function flipCard({ target: clickCard }) {
    if (!isPlaying || disableMultipleClick || countdownActive) {
        return; // Do nothing if the game is not started or card clicking
        is disabled
    }
}

```



```

    if (clickCard !== firstCard && !disableMultipleClick && timeLeft > 0
    && !clickCard.classList.contains('matched')) {
        if (!clickCard.classList.contains('flipping')) {
            playClickSound();
        }
        flips++;
        flipsCount.innerText = `Flips count: ${flips}`;
        clickCard.classList.add('flipping');
        if (!firstCard) {
            return firstCard = clickCard;
        }
        secondCard = clickCard;
        disableMultipleClick = true;

        let firstCardImage = firstCard.querySelector('.front img').src,
            secondCardImage = secondCard.querySelector('.front img').src;
        matchingCards(firstCardImage, secondCardImage);
    }
}

```

Matching Cards:

- `matchingCards(image1, image2)`: Checks if two flipped cards match, handles the match logic, updates the score, and manages unmatched card behavior.

```

function matchingCards(image1, image2) {
    if (image1 === image2) {
        matchCard++;
        consecutiveMatches++;
        score += consecutiveMatches;
        scoreDisplay.innerText = `Score: ${score}`; // Update score
display
        playMatchSound();
        if (matchCard === 8 && timeLeft > 0) {
            clearInterval(time);
            playVictorySound();
            let elapsedTime = Math.round((Date.now() - startTime) / 1000);

            endGame('Victory', flips, elapsedTime, score);
        }

        setTimeout(() => {

```

```

        firstCard.classList.add('matched');
        secondCard.classList.add('matched');
        firstCard.removeEventListener('click', flipCard);
        secondCard.removeEventListener('click', flipCard);
        setTimeout(() => {
            firstCard.style.visibility = 'hidden';
            secondCard.style.visibility = 'hidden';
            firstCard = secondCard = '';
            disableMultipleClick = false;
        }, 500);
    }, 500);
} else {
    consecutiveMatches = 0; // Reset consecutive matches
    setTimeout(() => {
        firstCard.classList.add('notMatchedShake');
        secondCard.classList.add('notMatchedShake');
    }, 400);

    setTimeout(() => {
        firstCard.classList.remove('notMatchedShake', 'flipping');
        secondCard.classList.remove('notMatchedShake', 'flipping');
        firstCard = secondCard = '';
        disableMultipleClick = false;
    }, 1200);
}
}
}

```

Game Control

Stop Game:

- `stopGame()`: Ends the game by stopping the timer, disabling card clicks, and displaying a game over message.

```

function stopGame() {
    clearInterval(time);
    isPlaying = false;
    disableMultipleClick = true;
    endGame('Game Over', flips, maxTime - timeLeft, score);
    playGameOverSound();

    // Disable card clicking
}

```

```

cards.forEach((card) => {
  card.removeEventListener('click', flipCard);
  card.style.visibility = 'visible';
});
}

```

Countdown Before Game Start:

- `startCountdown()`: Initiates a countdown before the game starts, displaying numbers from 3 to "Go!". And disable the buttons and click when the countdown is on

```

function startCountdown() {
  button.disabled = true; // Disable the button during countdown
  resetButton.disabled = true; // Disable the reset button during
countdown
  countdownActive = true; // Set countdown as active

  let countdown = 3;
  const countdownElement = document.querySelector('.countdown');
  countdownElement.innerText = countdown;
  countdownElement.style.display = 'block'; // Ensure countdown is
visible

  const countdownInterval = setInterval(() => {
    countdown--;
    if (countdown > 0) {
      countdownElement.innerText = countdown;
    } else if (countdown === 0) {
      countdownElement.innerText = 'Go!';
    } else {
      clearInterval(countdownInterval);
      countdownElement.style.display = 'none'; // Hide countdown
message
      button.disabled = false; // Enable the button after countdown
      // resetButton.disabled = false; // Enable the reset button
after countdown
      startGame(); // Start the game after countdown
      button.innerText = 'refresh'; // Change button text to
"refresh" after countdown finishes
      countdownActive = false; // Reset countdown active state
    }
  }, 1000);
}

```

Start Game:

- `startGame()`: Starts a new game, resetting necessary states and starting the timer.

```
function startGame() {
  resetGame(); // Initialize the game
  isPlaying = true;
  resetButton.disabled = false; // Enable reset button when game starts
  startTime = Date.now();
  time = setInterval(initTimer, 1000);
  disableMultipleClick = false; // Enable card clicking when game starts
  message.style.display = 'none'; // Hide the message when game starts
}
```

Toggle Button Text:

- `toggleText(event)`: Toggles the start and refresh button functionality based on the current game state.
- Also the reset button is disabled when the game end as to don't end the game 2 times even if the game didn't start .

```
//Function to toggle text and start countdown
function toggleText(event) {
  var text = event.textContent || event.innerText;
  if (!isWaitingForUsername) {
    if (text == 'start') {
      startCountdown(); // Start the countdown
    } else if (text == 'refresh') {
      startGame();
    }
  }
}

// Event listener for the button
button.addEventListener('click', () => toggleText(button));

// Add event listener for reset button to stop the game
resetButton.addEventListener('click', () => {
  if (!isWaitingForUsername) {
    stopGame();
    message.innerText = 'You ended the game';
    message.style.display = 'flex';
    resetButton.disabled = true; // Disable reset button when
displaying message
  }
});
```

```

    }
  });

message.addEventListener('click', () => {
  message.style.display = 'none';
});

// Event listener for the cards
cards.forEach((card) => {
  card.addEventListener('click', (event) => {
    if (!countdownActive && isPlaying) { // Check if the game is in
progress
      flipCard(event);
    }
  });
});
});

```

Leaderboard Functionality

Leaderboard Initialization:

- o `initializeLeaderboard()`: Ensures the leaderboard is initialized in local storage.

```

// Initialize leaderboard in local storage if it doesn't exist
function initializeLeaderboard() {
  if (!localStorage.getItem('leaderboard')) {
    localStorage.setItem('leaderboard', JSON.stringify([]));
  }
}

```

Update Leaderboard:

- `updateLeaderboard(playerName, flips, timeTaken, score)`: Updates the leaderboard with the player's score if it is higher than an existing score or adds a new entry.

```

function updateLeaderboard(playerName, flips, timeTaken, score) {
  let leaderboard = JSON.parse(localStorage.getItem('leaderboard'));
  let existingPlayerIndex = leaderboard.findIndex(entry => entry.name
=== playerName);

```

```

    // If the player exists in the leaderboard
    if (existingPlayerIndex !== -1) {
        // Update score only if the new score is higher
        if (score > leaderboard[existingPlayerIndex].score) {
            leaderboard[existingPlayerIndex] = { name: playerName, flips:
flips, timeTaken: timeTaken, score: score };
        }
    } else {
        // Add a new entry to the leaderboard
        leaderboard.push({ name: playerName, flips: flips, timeTaken:
timeTaken, score: score });
    }

    leaderboard.sort((a, b) => b.score - a.score);
    localStorage.setItem('leaderboard', JSON.stringify(leaderboard));
    displayLeaderboard();
}

```

Display Leaderboard:

- `displayLeaderboard()`: Displays the current leaderboard in the UI.

```

/ Display leaderboard
function displayLeaderboard() {
    let leaderboard = JSON.parse(localStorage.getItem('leaderboard'));
    const leaderboardElement = document.getElementById('leaderboard');
    leaderboardElement.innerHTML = '';
    leaderboard.forEach((entry, index) => {
        let listItem = document.createElement('li');
        listItem.innerText = `${index + 1}. ${entry.name} - Score:
${entry.score}, Flips: ${entry.flips}, Time: ${entry.timeTaken}s`;
        leaderboardElement.appendChild(listItem);
    });
}

```

Username Handling and Game End

Toggle Button State:

- `toggleButtonState(isGameOver)`: Toggles the state of control buttons based on whether the game is over.

```
function toggleButtonState(isGameOver) {
  if (isGameOver) {
    button.innerText = 'start'; // Change button text to "start" after
game over
    resetButton.disabled = false; // Enable reset button after game
over
  } else {
    button.innerText = 'refresh'; // Change button text to "refresh"
after countdown finishes
    resetButton.disabled = true; // Disable reset button until the
game is over
  }
}
```

End Game:

- `endGame(status, flips, elapsedTime, score)`: Ends the game, prompts for the player's username and updates the leaderboard.

```
function endGame(status, flips, elapsedTime, score) {
  isPlaying = false;
  disableMultipleClick = true;
  displayMessage(`${status}. Flips: ${flips}, Time Taken: ${elapsedTime}
seconds, Score: ${score}`);

  if
(!enableLeaderboardCheckbox.checked || enableLeaderboardCheckbox.checked) {
    usernameContainer.style.display = 'block'; // Show username input
field
    isWaitingForUsername = true;
    document.querySelector('.username-
container').style.display='flex';
    submitUsernameButton.onclick = function() {
      const playerName = usernameInput.value.trim();
      if (playerName) {
        updateLeaderboard(playerName, flips, elapsedTime, score);
        displayMessage(`${status}. Flips: ${flips}, Time Taken:
${elapsedTime} seconds`);
        usernameContainer.style.display = 'none'; // Hide username
input field
        isWaitingForUsername = false;
        document.querySelector('.username-
container').style.display='none';
      }
    }
  }
}
```

```

        document.querySelector('.message').style.display='none';
    } else {
        alert('Please enter a valid name to save your score.');
```

Additional Features

Sound Toggle:

- o toggleSound(): Toggles the sound on or off based on the checkbox state

```

// Function to toggle the leaderboard display based on checkbox state
function toggleLeaderboardDisplay() {
    const leaderboardContainer =
document.getElementById('leaderboardContainer');
    if (enableLeaderboardCheckbox.checked) {
        leaderboardContainer.style.display = 'block'; // Display
leaderboard if checkbox is checked
        displayLeaderboard(); // Update and display leaderboard
    } else {
        leaderboardContainer.style.display = 'none'; // Hide leaderboard
if checkbox is unchecked
    }
}
// Add event listener to the checkbox
enableLeaderboardCheckbox.addEventListener('change',
toggleLeaderboardDisplay);

// Initialize leaderboard on page load
initializeLeaderboard();
displayLeaderboard();

// Get the checkbox element
const enableSoundCheckbox = document.getElementById('enableSound');

// Function to toggle audio sound
```



```
function toggleSound() {
    clickSound.muted = !enableSoundCheckbox.checked;
    matchSound.muted = !enableSoundCheckbox.checked;
    victorySound.muted = !enableSoundCheckbox.checked;
    gameOverSound.muted = !enableSoundCheckbox.checked;
}

// Add event listener to the checkbox to toggle sound
enableSoundCheckbox.addEventListener('change', toggleSound);

// Call toggleSound function initially to set the initial state of the sound
toggleSound();
```

Overall Flow

The game operates in a clear and structured manner, guided by the user interactions and the game logic embedded within the functions and event listeners.

1. Game Start:

- Player clicks the start button.
- A countdown sequence begins, displaying "3", "2", "1", and then "Go!".
- The game state is reset, and the game begins, enabling card clicks and starting the timer.

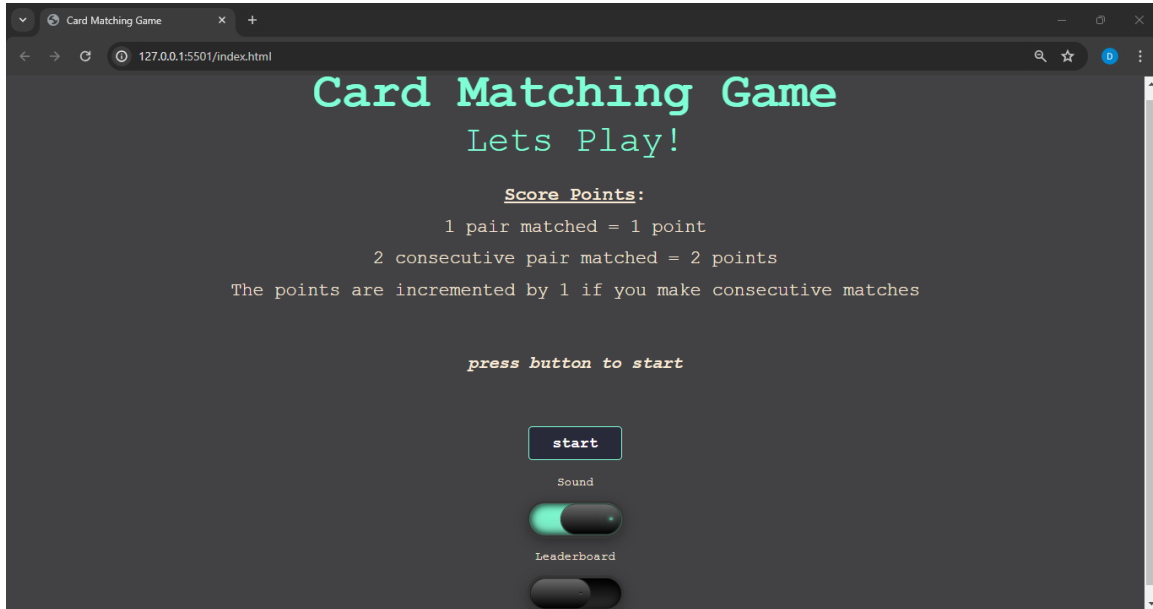
2. Card Flipping:

- Players click on cards to flip them.
- If two cards are flipped, the game checks for a match.
- Matching cards increase the score and are marked as matched.
- Unmatched cards are flipped back after a short delay.

3. Game End:

- The game ends either when all cards are matched or when the timer runs out or when the user reset.
- Players can submit their score to the leaderboard.
- The leaderboard is updated and displayed if the score is among the top entries.

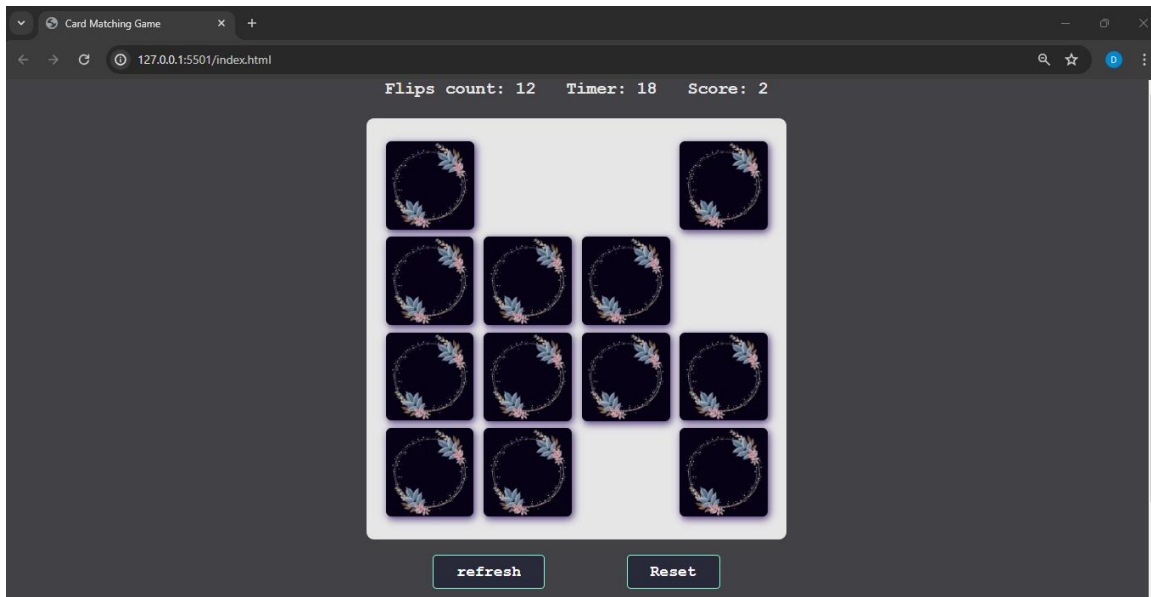
Screenshot :



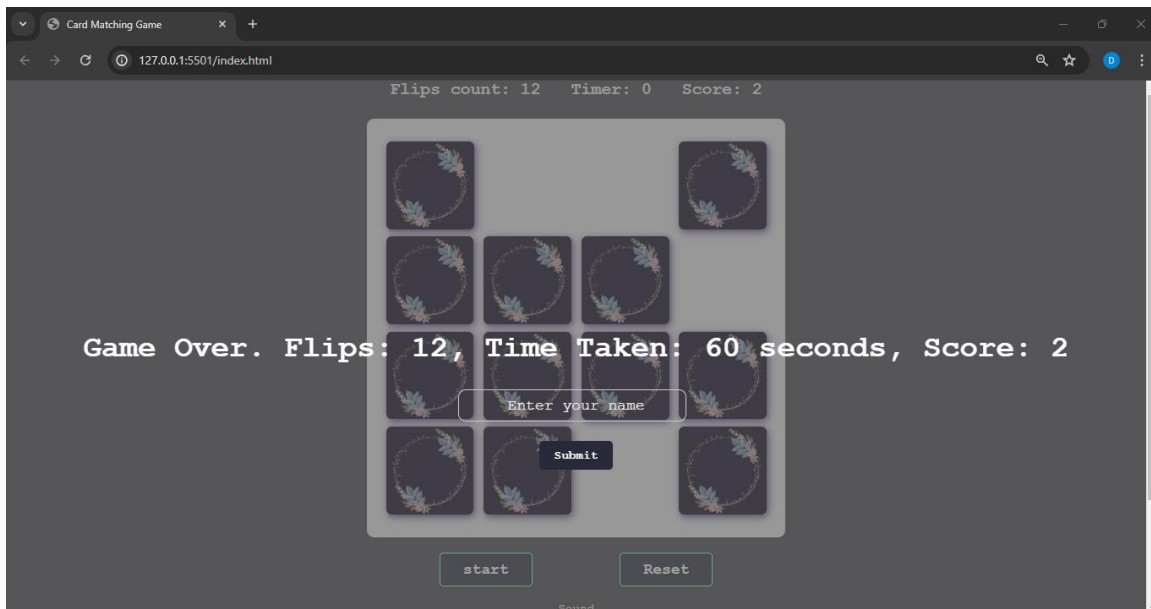
Count down when click start

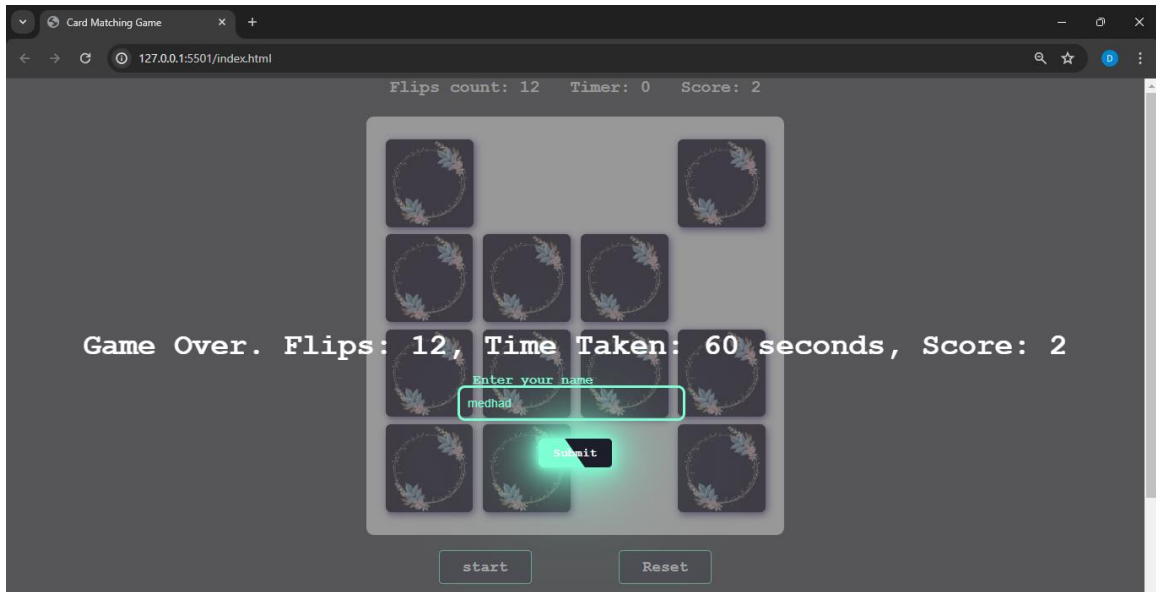


Started the game and make a match cards and excluded

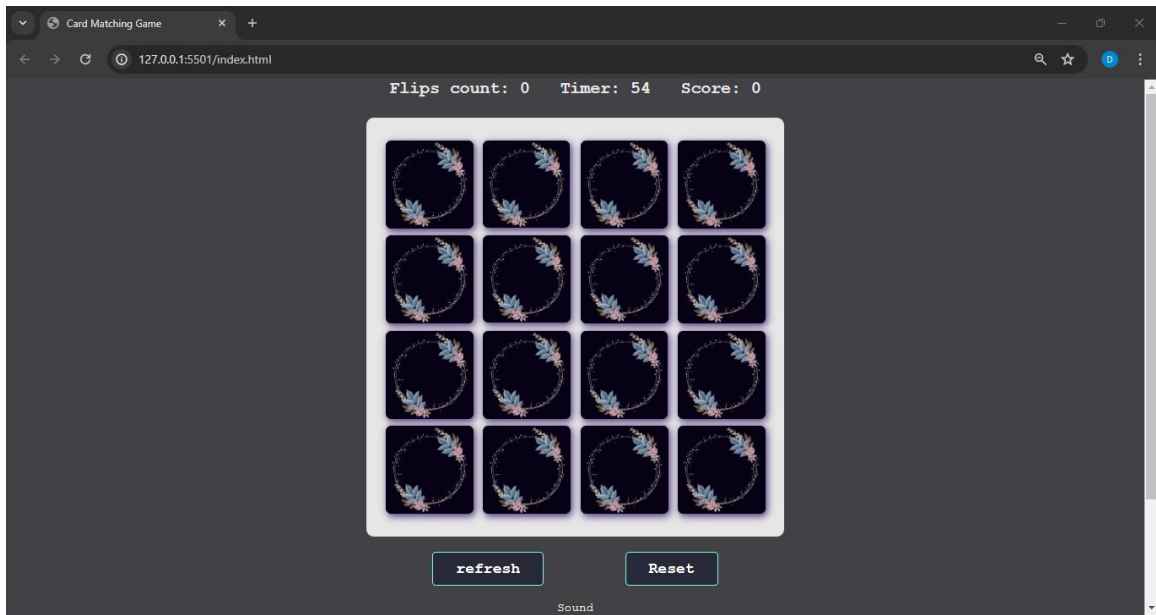


Lost so game over and the user have to enter his username to be able to update in the leaderboard and continue if he wants to play again

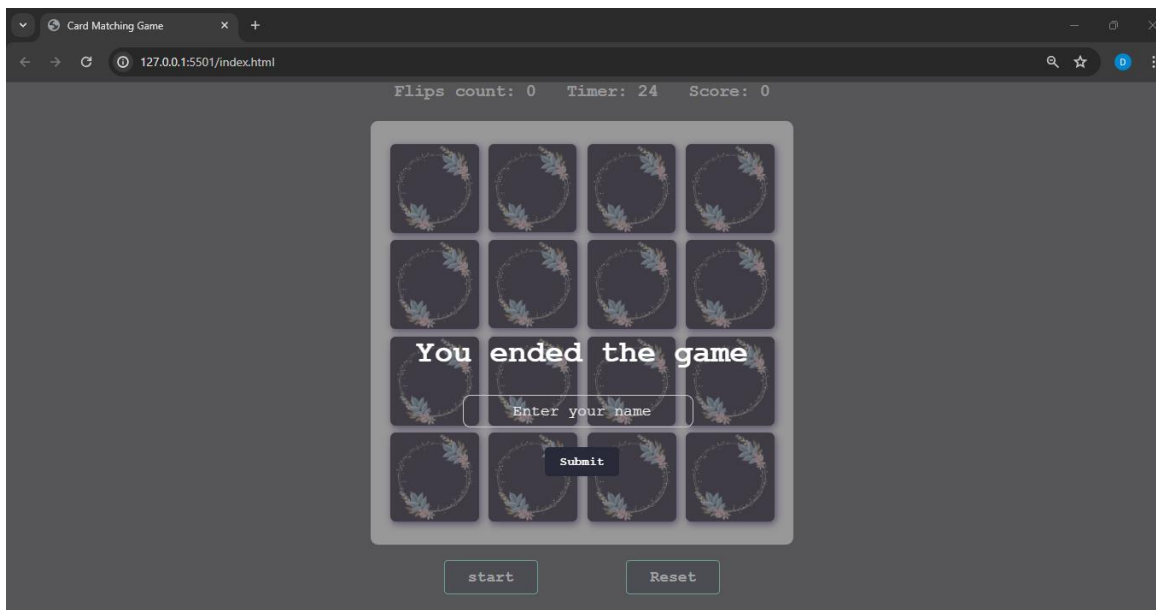




Can enable leaderboard and the information is added also can enable or disable the sound



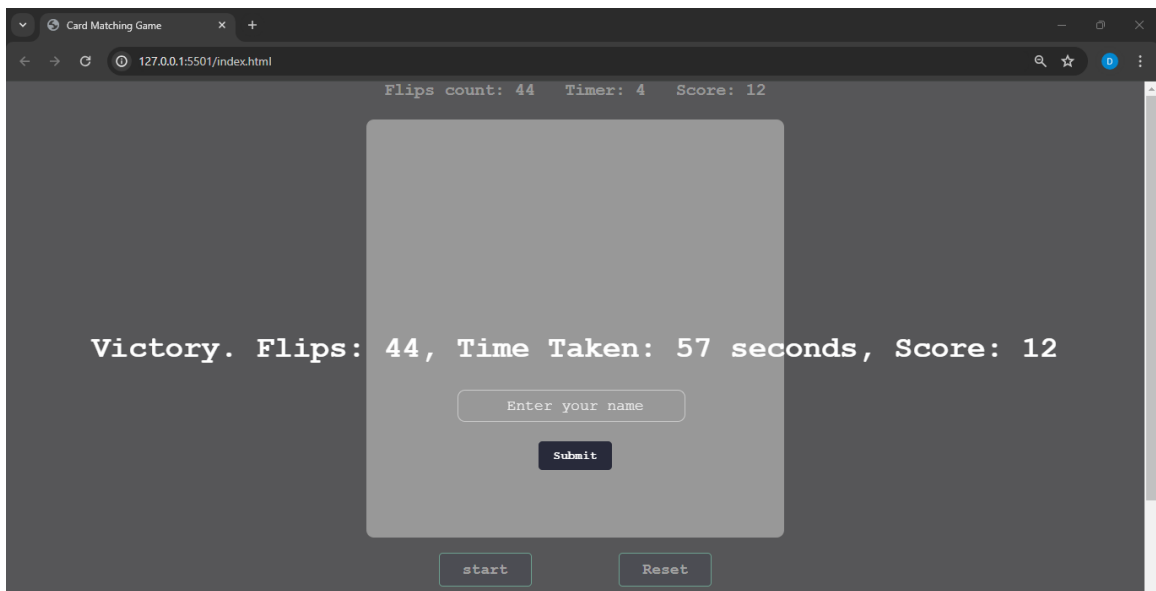
Started again and cards are flipped and shuffle



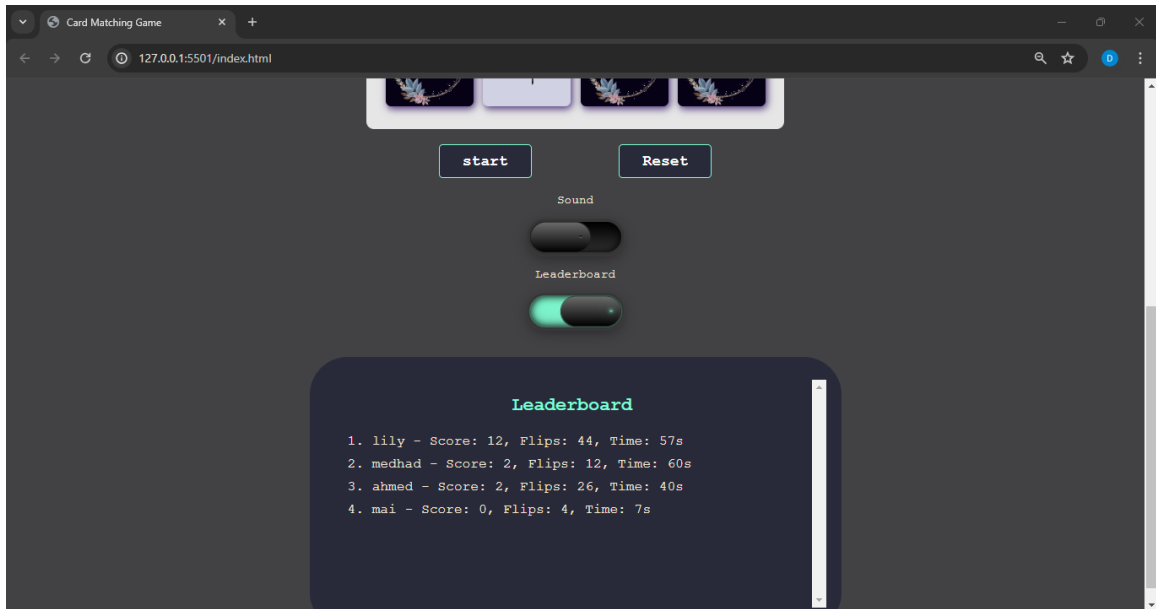
Clicked reset and the game is ended and enter username



Information updated



Win the game , displayed a message and cards are excluded



When played again the score is updated with the higher score as in ahmed

Link for video

<https://drive.google.com/drive/folders/1RojyrHNvrlIfkY4b4p26yBtlxTILro2o?usp=sharing>