AMOR A PRIMERA CARTA

Manual and game developed:
Yazu Bautista
Design and layout:
Derek RevillA
TRANSLATED BY:
ALEXANDRA DIAZ

GAME MANUAL: LOVE AT FIRST CARD

1. Introduction

The game "Love at First Card" is a project developed with HTML, CSS, and JavaScript, where the player's objective is to find pairs of matching cards.
When the game starts, all the cards are shown face down.
The player must click on two cards at a time to attempt to discover an identical pair of heart icons 💜💙💘💞💛💗❣️🤎.
If the cards match, they will remain revealed; if not, they will be flipped back over after one second.
The game tests the player's visual memory and concentration.

2. Required Files

The project uses three main files:
● HTML (uno.html) $\rightarrow$ Base structure of the game. It contains the board,
the button, and the connection to the other files.
● CSS (uno.css) $\rightarrow$ Defines the visual styles: colors, positions, typography,
sizes, and board organization.
● JavaScript (uno.js) $\rightarrow$ Controls the game logic: card shuffling, pair
detection, reset, and click handling.
No external images are needed since the game uses emojis to represent the cards. The game's
background is a solid color ($\text{\#2c3e50}$) that provides an elegant and modern
ambiance.

3. HTML Structure - uno.html

This file defines the skeleton of the game and links the styles and the logic.
Detailed Explanation:

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

● Declares that the document is HTML5 and is in English.

```
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>AMOR A PRIMERA CARTA</title>
  <link rel="stylesheet" type="text/css" href="uno.css">
</head>
```

● <meta charset="utf-8"> allows special characters and emojis to display correctly.
● <meta name="viewport"...> ensures that the game looks good on mobiles and tablets.
● <link> connects to the external styles file uno.css.

```
<body>
  <h1> AMOR A PRIMERA CARTA </h1>
  <div class="game-board"></div>
  <button id="reset-button"> Reset Game </button>
  <script src="uno.js"></script>
</body>
</html>
```

● <h1> displays the main title of the game.
● will be the container where JavaScript will dynamically
insert the cards.
● <button id="reset-button"> allows the game to be reset.
● <script src="uno.js"> links the game logic file.

4. Visual design - uno.css
This file gives shape, color, and order to all the game elements.
General Structure:

```
body {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    height: 100vh;
    background-color: #2c3e50;
    color: white;
    font-family: Arial, sans-serif;
}
```

● display: flex allows all elements in the body to be centered.
● flex-direction: column arranges them in a column (title, board, button).
● align-items: center and justify-content: center center the content horizontally and
vertically.
● height: 100vh occupies the entire viewport height.
● background-color: #2c3e50 provides an elegant dark tone.
● color: white creates contrast with the background.
● font-family: Arial maintains readability.

Title:

```
h1 {
    margin-bottom: 20px;
}
```

● Separates the title from the board so they aren't touching.

Game Board:

```
.game-board {
    display: grid;
    grid-template-columns: repeat(4, 100px);
    grid-gap: 10px;
}
```

● CSS Grid is used to organize the cards in a grid.
● grid-template-columns: repeat(4, 100px) $\rightarrow$ creates 4 columns of 100px
each.
● grid-gap: 10px $\rightarrow$ leaves space between the cards34. This forms a 4x4 board (16 cards).

Cards:

```css
.card {
    width: 100px;
    height: 100px;
    background-color: #ecf0f1;
    display: flex;
    align-items: center;
    justify-content: center;
    font-size: 24px;
    cursor: pointer;
    border-radius: 8px;
}
```

● Defines the uniform size of each card (100x100px).
● background-color: #ecf0f1 $\rightarrow$ a neutral color while it is "hidden".
● display: flex; align-items: center; justify-content: center $\rightarrow$ centers the
emoji on the card.
● font-size: 24px $\rightarrow$ visible icon size.
● cursor: pointer $\rightarrow$ changes the cursor to a hand, indicating it is clickable.
● border-radius: 8px $\rightarrow$ rounded edges for a softer style41
.

Flipped Card:

```css
.card.flipped {
    background-color: #3498db;
    color: white;
}
```

● When the card is "flipped", the background changes to blue ($\text{\#3498db}$) and the text (emoji) turns white, simulating the uncovering action.

Reset Button:

```css
#reset-button {
    margin-top: 20px;
    padding: 10px 20px;
    font-size: 16px;
    cursor: pointer;
    border: none;
    border-radius: 5px;
    background-color: #e74c3c;
    color: white;
}
```

● Gives the button a modern design.
● background-color: #e74c3c $\rightarrow$ an intense red that stands out against the
background.
● cursor: pointer indicates interactivity.
● border-radius and padding soften its appearance.

5. Game logic - uno.js

All the interactivity , such as clicks, pairs, reset, and card status, is controlled here.

Variable Declaration:

```
const cards = [
  '💜', '💜', '💙', '💙', '💝', '💝',
  '💞', '💞', '💛', '💛', '💗', '💗',
  '❣️', '❣️', '🤎', '🤎',
];
```

● Creates an array with the icons duplicated, forming 8 pairs.

```
let firstCard = null;
let secondCard = null;
let lockBoard = false;
```

● These variables control the current state:
○ firstCard and secondCard: store the flipped cards.
○ lockBoard: prevents the player from clicking while the cards are flipping back over.

Card Shuffling 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]];
  }
}
```

● Fisher-Yates algorithm is used, which randomly shuffles the array elements,
ensuring a different game every time.

Creating the Board:

```
function createBoard() {
  const gameBoard = document.querySelector('.game-board');
  shuffle(cards);
  cards.forEach(card => {
    const cardElement = document.createElement('div');
    cardElement.classList.add("card");
    cardElement.dataset.icon = card;
    cardElement.textContent = '';
    cardElement.addEventListener('click', flipCard);
    gameBoard.appendChild(cardElement);
  });
}
```

● Generates all the cards visually:
○ Creates a for each card.
○ Uses dataset.icon to store its value without displaying it.
○ Adds the click event that allows the card to be turned over (flipped).

Flipping a Card:

```
function flipCard() {
  if (lockBoard) return;
  if (this === firstCard) return;

  this.classList.add('flipped');
  this.textContent = this.dataset.icon;

  if (!firstCard) {
    firstCard = this;
    return;
  }

  secondCard = this;
  checkForMatch();
}
```

● Controls the flip:
○ If the board is locked (lockBoard), the player cannot make a move.
○ The card is "flipped" by showing its emoji (dataset.icon).
○ When two cards are flipped, it calls checkForMatch().

Pair Check:

```
function checkForMatch() {
  if (firstCard.dataset.icon === secondCard.dataset.icon) {
    disableCards();
  } else {
    unflipCards();
  }
}
```

● Checks if the two cards are equal:
○ If they match $\rightarrow$ they are disabled.
○ If not $\rightarrow$ they are flipped back over.

Disabling Matched Cards:

```
function disableCards() {
  firstCard.removeEventListener('click', flipCard);
  secondCard.removeEventListener('click', flipCard);
  resetBoard();
}
```

● The correct cards can no longer be clicked, simulating that they remain "uncovered".

Flipping Incorrect Cards Back:

```
function unflipCards() {
  lockBoard = true;
  setTimeout(() => {
    firstCard.classList.remove('flipped');
    secondCard.classList.remove('flipped');
    firstCard.textContent = '';
    secondCard.textContent = '';
    resetBoard();
  }, 1000);
}
```

● Uses setTimeout to keep the incorrect cards visible for 1 second before hiding them again. During that time, lockBoard = true blocks new clicks.

State Reset:

```
function resetBoard() {
  [firstCard, secondCard, lockBoard] = [null, null, false];
}
```

● Clears the variables to allow the game to continue correctly.
Full Game Reset:
● When the button is pressed, the board is emptied and re-created with the cards

shuffled again.

```
document.getElementById('reset-button').addEventListener('click', () => {
  document.querySelector('.game-board').innerHTML = '';
  createBoard();
});
createBoard();
```

Conclusion

The game "Love at First Card" combines visual simplicity with clear and efficient logic.

Its flexible design allows it to adapt to any screen and is ideal for educational or

demonstrative programming projects. Every style value and every function has a precise

reason that ensures the game is intuitive, functional, and aesthetically pleasing.