

## > **Ficha Prática Nº3 (Jogo de Memória – Baralhar tabuleiro de jogo)**

Esta ficha, tem como objetivo implementar as funções necessárias para **baralhar o tabuleiro de jogo, alterar os logotipos e efetuar a rotação de cartas após serem clicadas**. O HTML e o CSS necessário à resolução da ficha, já se encontra implementado e fornecido, não devendo nenhum dos ficheiros ser alterados.

O resultado da ficha apresenta-se na figura seguinte, o qual se pretende apresentar as cartas baralhadas sempre que se inicia um jogo e virar as cartas quando se clica em cada uma.



Figura 1 – Jogo de Memória em JavaScript – Imagens da aplicação

## > **Apoio à resolução da ficha:**

**a.** A sintaxe genérica para criação de um **array** literal com valores é:

```
let nomeArray = [item1, item2, ...]
```

- Os elementos de um **array** podem ser alterados da seguinte forma:

```
nomeArray[0] = 'novoItem';  
nomeArray[1] = 23;
```

- Um **array** é um tipo especial de *Objecto* em *JavaScript*. Logo, como objeto, inclui um conjunto de propriedades e métodos que facilitam o seu acesso e sua manipulação.

Apresentam-se de seguida alguns exemplos:

```
let dimArray = nomeArray.length;
let arraySorted = nomeArray.sort(); // Ordena por ordem alfabética
nomeArray.push("NovoElemento"); // Adiciona no fim
nomeArray[dimArray] = 'NovoElemento';
nomeArray.pop(); // Remove o último
nomeArray.shift(); // Remove o primeiro
nomeArray = nomeArray.concat(['novo1', 'novo2']);
nomeArray = [...nomeArray, 'novo1', 'novo2'];
nomeArray.push(...nomeArray);
nomeArray = nomeArray.slice(0, 4); // Devolve elementos entre índice 0 e 4
nomeArray.splice(1, 1, 'novo1', 'novo2'); // Inserir elementos
```

**b.** A sintaxe genérica para definir um **for..of** é a seguinte:

```
for (variavel of iteravel) {
    //... código ser executado
}
```

**c.** A sintaxe genérica para definir um **forEach** é a seguinte:

```
elementos.forEach(function(elemento, index, arr)) {
    //...
});
> function – função a ser executada por cada elemento
> index – opcional, índice do elemento corrente
> arr – opcional, array do elemento corrente
```

**d.** O código abaixo apresenta um trecho de código HTML no qual existem **atributos data**. Os atributos data permitem adicionar informação adicional às tags HTML. Não são específicas do HTML5, mas os atributos **data-\*** podem ser usados em todos os elementos HTML. No contexto da ficha, é utilizado um atributo data para especificar qual é o logotipo da carta.

```
<div class="card" data-logo="javascript">
    
    
</div>
```

Para obter/atribuir o valor de/a um atributo **data**, pode-se recorrer à propriedade **dataset** como se apresenta no exemplo seguinte, o qual obtém/atribui o valor do/ao atributo **data-logo**:

```
let logotipo = card.dataset.logo;
card.dataset.logo = 'react'
```

- e.** Quando se adiciona um *Event Listener* com recurso ao `addEventListener`, o objeto que recebe uma notificação quando um evento do tipo especificado ocorre é o **listener**. Para identificar o elemento, pode-se recorrer à propriedade **currentTarget**. Além disso, a palavra-chave **this** permite referenciar o elemento do qual a espera de evento foi disparada, como quando é usado um manipulador genérico para uma série de elementos similares. Resumindo, o valor **this** permite obter "*qualquer objeto em que uma determinada função seja executada*", dependendo de como a função é chamada e varia se é usado o modo restrito ou não. Como exemplo:

```
const button = document.querySelector(".elemento");
button.addEventListener("click", funcaoManipulaClick);

function funcaoManipulaClick() {
  console.log("Botão Clicado!");
  this.style.border = "5px blue solid";
}
```

```
button.addEventListener("click", function () {
  funcaoManipulaClick(this);
});

function funcaoManipulaClick(elem) {
  console.log("Botão Clicado!");
  elem.style.border = "5px blue solid";
}
```

```
button.addEventListener("click", function (e) {
  funcaoManipulaClick(e.currentTarget);
});

function funcaoManipulaClick(elem) {...}
```

## > Preparação do ambiente

- a. Descompacte o ficheiro **ficha3.zip**.



**Os alunos que concluíram a resolução da ficha anterior, devem continuar nesse trabalho.** Os

restantes alunos podem resolver esta ficha prática, tendo como base o código fornecido em ficha3.

- b. Inicie o *Visual Studio Code* e abra a **pasta no workspace**.

Visualize a página **index.html** no browser (recorra à extensão "*Live Server*"), no qual terá o aspeto da figura 2.

- c. Passos gerais a implementar na ficha:

- Criar tabuleiro com logotipos diferentes após iniciar jogo (Parte I > 1);
- Limitar o tabuleiro com logotipos pares, portanto, 3 pares (Parte I > 2);
- Virar carta após ser clicada (Parte II)
- As partes seguintes (III, IV e V), para explorar em casa, detalham os passos que permitem implementar alguns comportamentos usando técnicas alternativas.



Figura 2 - Ficha 2

## > Explicação de algum código HTML e regras CSS

- a. O comportamento que especifica o contorno quando o rato passa em cima de uma carta (figura ao lado), está implementado apenas com uma regra CSS, sem recorrer a qualquer JavaScript. O CSS encontra-se no ficheiro **style.css** e o seletor que implementa esse comportamento é o **.card:hover**.



Figura 3 - Carta Seleccionada

- b. O código abaixo apresenta um trecho de código HTML no qual permite apresentar no browser uma carta, código este que pode ser visto no ficheiro **index.html**. Como pode verificar, cada carta é composta por duas imagens como o exemplo abaixo: a imagem **ls.png** e, neste trecho apresentado, a imagem **react.png**. Além disso, são usadas as **classes card-back** e **card-front** que especificam as propriedades CSS para que as duas imagens fiquem sobrepostas uma na outra, em que a classe **card-front** inclui uma rotação 180° no eixo dos Y à imagem de forma a permitir o efeito de rotação da carta, quando houver um clique.

```
<div class="card" data-logo="javascript">
  
  
</div>
```

## Parte I – Baralhar as Cartas do Tabuleiro

**1>** Pode-se recorrer a várias técnicas para baralhar as cartas existentes no tabuleiro de jogo, **uma recorrendo ao CSS**, no qual se altera a propriedade *order* do *grid layout*, e outra, por exemplo, **com recurso à manipulação de arrays**. A secção “Parte III” da ficha, apresenta uma versão que recorre ao CSS para baralhar as cartas. No contexto da aula, será abordada uma versão **com recurso à manipulação de arrays**.

**a.** Como pode ver quando inicia o jogo, como apresenta a Figura 6, todas as cartas apresentam o logotipo “Linguagens Script?”. De forma a ser possível ver o logotipo que se encontra por trás de cada carta, implemente duas funções a **showCards** e **hideCards**.

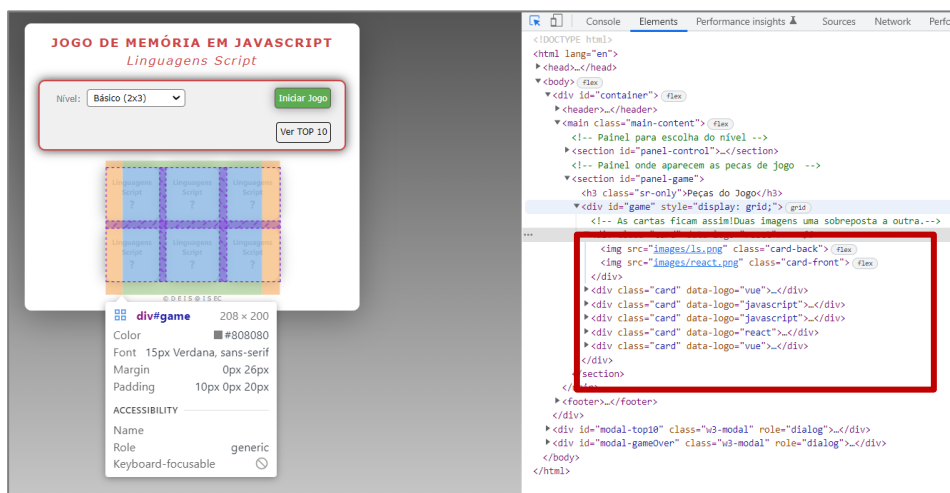


Figura 4 - Jogo de Memória – Cartas

- A função **showCards** deverá efetuar a rotação de todas as cartas existentes no tabuleiro. Para isso, **especifique**:
  - uma variável **cards**, com *scope* global, devendo ser inicializada com a referência de todos os elementos html especificados com a classe **.card**, que se encontram na zona do **panelGame**. Para isso use o método **querySelectorAll**. Note que a variável **cards** será um array de elementos, i.é., um array com todas **as cartas existentes** e visíveis no browser.
  - Para implementar a função **showCards**, recorra, por exemplo, ao **for...of**, e percorra o array **cards** **anteriormente definido** e adicione a classe **flipped** a cada um dos elementos existentes. Invoque esta função sempre que o jogo começa, na função **startGame**
  - Para implementar a função **hideCards** remova a classe **flipped** a cada um dos elementos existentes no array **cards**. Invoque esta função sempre que o jogo termina, na função **stopGame**.



Figura 5 - Cartas Viradas

Visualize o jogo no browser e certifique-se que ao carregar no botão **Iniciar/Terminar jogo**, as cartas efetuam uma rotação e que estas têm logotipos diferentes a cada jogo iniciado.

- b.** O próximo passo será então sim, baralhar o tabuleiro. Assim, declare o array **cardsLogos** (veja a secção **Apoio à resolução da ficha**) com os seguintes valores:

- angular
- bootstrap
- html
- javascript
- vue
- svelte
- react
- css
- backbone
- ember

- c.** Adicione a função **shuffleArray** apresentada abaixo, função esta que permite baralhar os elementos de um *array*, passado por parâmetro, e retorna o *array* já baralhado. **Não deve efetuar qualquer alteração a esta função.**

```
// Algoritmo Fisher-Yates - Algoritmo que baralha um array.
const shuffleArray = array => {
  for (let i = array.length - 1; i > 0; i--) {
    const j = Math.floor(Math.random() * (i + 1));
    const temp = array[i];
    array[i] = array[j];
    array[j] = temp;
  }
}
```

- d.** Na função **startGame** invoque a função **shuffleArray(cardsLogos)**.

Verifique o estado do *array*, imprimindo na consola o *array* **antes** e **depois** da chamada à função *shuffleArray* e verifique se, de facto, o *array* **cardsLogos** passou a ter os seus *items* com uma ordem diferente (baralhados), como se apresenta na figura 4.

```
console.table(cardsLogos)
```



**Note que, apesar do array estar baralhado na consola, ainda não aconteceu o mesmo no browser!**

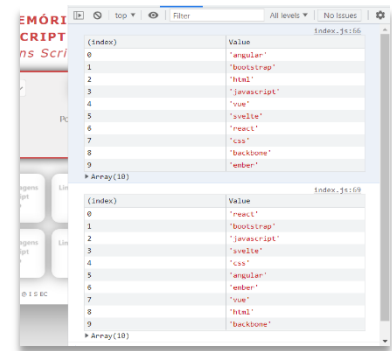


Figura 6 - Apresentação do array na consola

- e. Com o *array* **cardsLogos** baralhado, pretende-se alterar efetivamente as cartas a serem apresentadas no browser.

Nesse sentido, analise o trecho de código HTML apresentado abaixo, com especial atenção aos elementos destacados, e verifique que:

- A carta apresenta o logotipo do *javascript* quando o atributo **data-logo** é **javascript**, bem como o **src** (nome) do ficheiro é **javascript.png**

```
<div class="card" data-logo="javascript">
  
  
</div>
```



- Na pasta **images** fornecida com a ficha, existem várias imagens em que o nome do ficheiro é igual ao nome dos *items* do *array* **cardsLogos**. Este comportamento é propositado de forma a associar de forma simples um item ao respetivo ficheiro, portanto o item *javascript* tem a imagem *javascript.png*.

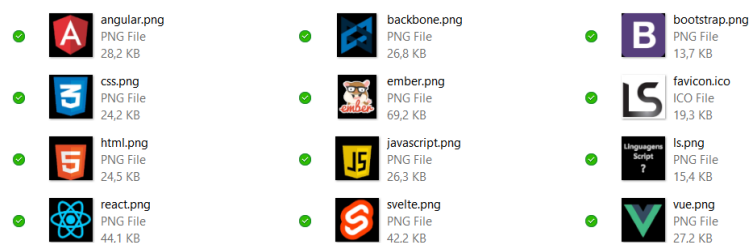


Figura 7- Imagens com logotipos

Pretende-se neste momento alterar as cartas a apresentar no painel de jogo de forma aleatória.

Tendo em consideração os passos anteriores, implemente agora os seguintes passos:

f. Para que sejam apresentados outros logotipos:

- Implemente, na função `startGame`, um **ciclo com recurso ao for...of**, que percorra todas as cartas (todos os elementos) existentes no array `cards`, de forma a alterar os dados da carta de acordo com os *items* do array `cardsLogos`. Isto é, note que o primeiro elemento do array `cards` é o react, logo:

```
<div class="card" data-logo="react">
  
  
</div>
```



Se o `cardsLogos` estiver com os valores `[html, 'react', 'backbone', 'ember', 'svelte', 'css', ...]`, a primeira carta a ser apresentada deverá ser o `html`, e, portanto, os dados deverão ser alterados para:

```
<div class="card" data-logo="html">
  
  
</div>
```

**NOTE QUE:** O atributo `src` tem de ser alterado com o nome do ficheiro pretendido no elemento com class `card-front`. Como existem vários elementos com essa classe, certifique-se que está a obter o elemento correto (dentro do ciclo e da carta em questão), no qual é necessário alterar o atributo.

- Implemente comportamento do ponto anterior, mas agora recorrendo a um ciclo **foreach**.
- Visualize no browser o comportamento, que deverá ser semelhante ao das figuras abaixo.



Figura 8 - Cartas com diferentes logotipos

g. Como pode verificar, **todas as cartas têm diferentes logotipos e não é esse o objetivo do jogo de memória**, no qual devem existir pares de logotipos. Será esse o passo seguinte.



**2>** Altere o código anteriormente implementado, de forma que o tabuleiro **panelGame** fique com o aspeto da figura 9, isto é, **cada carta tem o par**. Para implementar esta situação, uma das formas poderá ser da seguinte forma:

- **Especificar** o array **newCardLogos** que deve conter apenas **os 3 primeiros elementos do cardLogos**, após este estar baralhado (caso contrário serão sempre os mesmos logotipos usados). *Exemplo* : `newCardLogos = ['ember', 'react', 'javascript']`
- **Duplicar o novo array**. Consulte a secção de apoio: método *splice*, método *slice*, método *push* ou operador *spread* ...

```
newCardLogos=['ember','react','javascript','ember','react','javascript']
```

- **Baralhar todo o array otido**.

```
newCardLogos=['ember','javascript','javascript','react','ember','react']
```



Figura 9 - Cartas Baralhadas

## Parte II – Rotação da carta após clique

- 3>** Nesta fase, pretende-se especificar o código necessário para que, ao clicar numa carta, a mesma efetue uma rotação de forma a mostrar o logotipo.
- a.** Por forma a implementar este comportamento, coloque em comentário a invocação da função `showCards` existente em `startGame`, que foi implementada apenas para verificar se o processo de baralhar as cartas decorria de acordo com o pretendido.
  - b.** Implemente um *action listener* para cada uma das cartas existentes no tabuleiro e a função `flipCard` (a implementar de seguida) sempre que haja um clique. Para isso, **especifique um ciclo** que adicione o *listener* a cada uma das cartas (ou adicione este comportamento ao ciclo já existente no `startGame`).
  - c.** Por fim, implemente a função `flipCard()` cujo objetivo é efetuar a rotação da carta clicada. Para aceder à carta clicada na função `flipCard`, recorra à keyword `this`.
  - d.** Verifique no browser a rotação das cartas, sempre que existe um clique sobre ela.

## > Parte III, Parte IV e Parte V – Explorar Implementações alternativas

### Parte III – Baralhar cartas com propriedade order do CSS

4> Como referido anteriormente, é possível baralhar as cartas recorrendo à propriedade **order** do *grid layout* (CSS). Para isso, comente a parte de ordenação implementada nas secções anteriores e considere os seguintes passos:

- a. A variável **cards** já se encontra declarada anteriormente e esta referencia todos os elementos especificados com a classe **.card**, que se encontram no **panelGame**. Abaixo apresentam-se imagens onde pode ver o código html e CSS de como as cartas estão a ser especificadas.

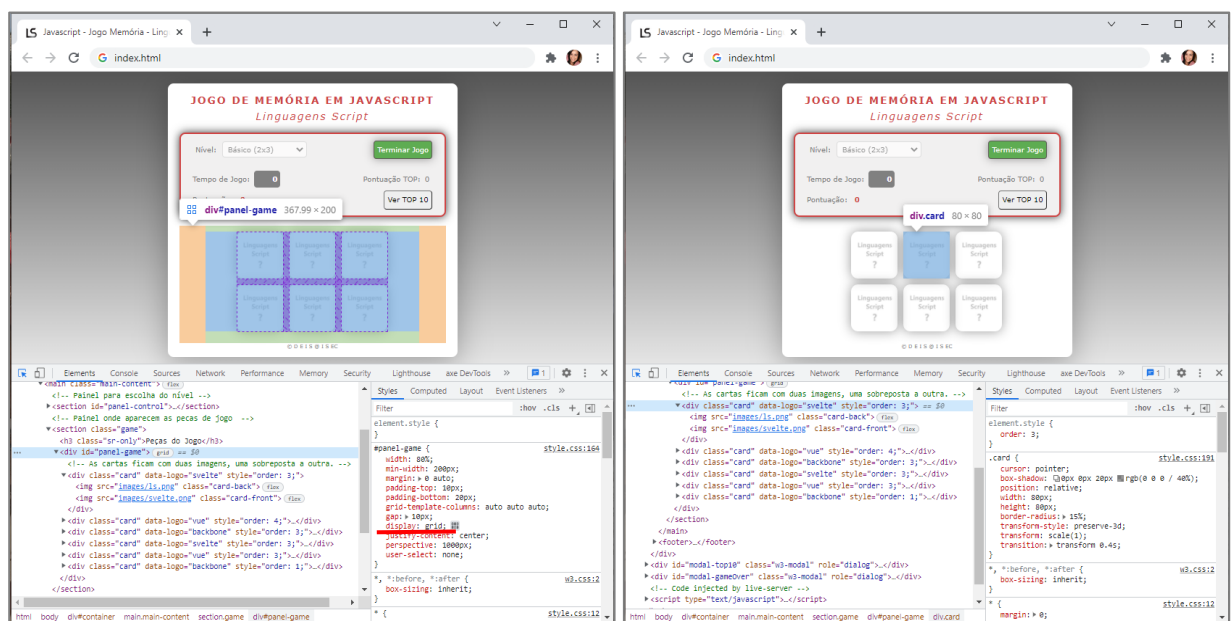


Figura 10 - Panel-game HTML + CSS

- b. Como pode verificar nas imagens anteriores, as cartas estão distribuídas no **panelGame** com recurso ao **grid layout** (ver ficheiro CSS). Por omissão, sem qualquer ordem especificada, a carta será colocada pela ordem especificada no HTML, sentido esquerdo-direito, cima-baixo. Ao atribuir um valor numérico à propriedade **order**, é possível alterar a posição/ordem do elemento. Por exemplo, ao especificar o estilo **order:2** a um elemento, o item será o segundo item ao longo do eixo principal. Nesse sentido, uma das formas para baralhar as cartas, será com recurso a essa propriedade, que deve ser aplicada a todas as cartas, de forma aleatória.

Para efetuar este processo, implemente os seguintes passos, na função **startGame()**:

- O código seguinte permite obter um valor aleatório entre 1 e o número de cartas existentes, que será 6 neste caso.

```
const randomNumber = Math.floor(Math.random() * cards.length) + 1;
```

- Com recurso ao **for... of** ou o **forEach**, percorra todos as cartas existentes (obtidas em **b.**) e aplique a propriedade **order**, especificando o valor obtido em **randomNumber**. Note que, o valor aleatório também deve ser obtido em cada iteração do ciclo.
- Verifique no browser que obteve o comportamento desejado.

### Parte III – Alterar o contorno por Javascript

**5>** Como referido anteriormente, o comportamento de colocar o contorno na carta quando o rato passa em cima de uma carta, está implementado sem recorrer a qualquer *JavaScript*. Apenas foi usada uma regra CSS, que pode encontrar no ficheiro *style.css*.

Pretende-se nesta secção, efetuar o mesmo comportamento, mas agora recorrendo ao *JavaScript*. Assim, implemente os seguintes passos:



Figura 11 - Carta Seleccionada

**a.** No ficheiro CSS, coloque em comentário a regra, e crie uma classe **cardHover** como aqui apresentada →

```
.cardHover {
  border: 2px solid var(--globalColor);
  box-shadow: var(--boxshadow0);
}
/* .card:hover {
  border: 2px solid var(--globalColor);
  box-shadow: var(--boxshadow0);
} */
```

**b.** Implemente os *action listeners* necessários de forma a que aplique a classe **.cardHover** quando o rato passa por cima da carta, e remova quando o rato sai.

- **mouseover** - adiciona a class **cardHover**
- **mouseout** - remove a class

**c.** Verifique no browser se comportamento pretendido se mantém.

### Parte IIV – Melhorias no código (DRY Principle e Delegação de Eventos)

**6>** Analise todo o código implementado e verifique se existe código duplicado que possa ser eliminado ou modificado, de forma a ficar mais eficiente.

Algumas sugestões:

- a.** Reduzir o número de ciclos dentro da função **reset()** quando tal for possível;
- b.** Altere o código das funções anónimas de forma a usar a sintaxe de *arrow functions*;

Altere o código referente aos *event listeners* para uma forma mais eficiente, usando o método de “delegação de eventos”, em vez de estar a adicionar um evento para cada carta.