

## > Ficha Prática Nº3 (Jogo de Memória – Baralhar painel do jogo)

### Notas:

- Os alunos **não devem alterar** o documento HTML nem os ficheiros de estilos existentes, de forma a seguirem o propósito da ficha.
- **Não devem remover** a instrução `'use strict'` que se encontra no topo do ficheiro `index.js` de forma a que seja usada na implementação, uma variante mais restrita do *JavaScript*.
- **Esta ficha, tem como objetivo implementar as funções necessárias para baralhar o tabuleiro de jogo, alterar os logotipos e efetuar a rotação de cartas quando se clica na carta.** O HTML e o CSS necessário à resolução desta ficha, já se encontra implementado e fornecido no ficheiro em anexo.
- Recomenda-se a consulta das dicas, durante a resolução, apresentadas na seção **Dicas**.
- O resultado final 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

## > Dicas para 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. Apresenta-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 dentro do array
```

- 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. Efetue o download e descompacte o ficheiro **ficha3.zip** disponível no *inforestudante*.

**NOTA:** Os alunos que concluíram a resolução da ficha 2, devem usar essa versão, devendo apenas substituir o ficheiro `index.html` e ficheiro `style.css`, de forma a que as cartas fiquem com o aspeto desejado da figura 2 ao iniciar o jogo.



- b. Inicie o *Visual Studio Code*, abra a **pasta no workspace** e 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 painel com logotipos diferentes apos iniciar jogo (Parte I > 1);
  - Limitar painel a ter logotipos pares, portanto, 3 pares (Parte I > 2);
  - Virar carta quando se clica numa carta (Parte II)

## > Explicação de algum código HTML e regras CSS

- d. 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 2 - Carta Seleccionada

- e. 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

**1>** Pode-se recorrer a várias técnicas para baralhar as cartas existentes no painel 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**. As dicas para implementar com recurso ao CSS encontram-se na secção “Parte III”. No contexto da aula, será abordada uma versão com recurso à manipulação de *arrays*. Para isso implemente os seguintes passos:

**a.** Declare um array **cardsLogos** com os seguintes valores:

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

**b.** 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;
  }
}
```

**c.** Dentro da função **startGame** invoque a função **shuffleArray(cardsLogos)**.

**d.** De forma a verificar o estado do *array*, imprima 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 exemplo, como se apresenta na figura .

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

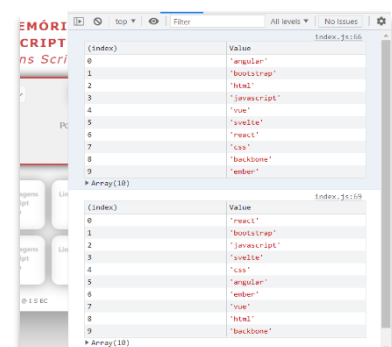


Figura 3 - Apresentação do array na consola

e. Com o array **cardsLogos** baralhado, pretende-se alterar efetivamente as cartas 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">
  
  
```



- 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 *react* tem a imagem *react.png*.

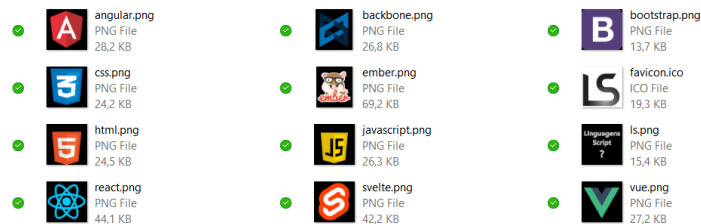


Figura 4- Imagens com logotipos

f. Pretende-se neste momento alterar as cartas a apresentar no painel de jogo de forma aleatória. Tendo em consideração os passos anteriores, implemente os seguintes passos:

- Especifique a 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 recorra ao método **querySelectorAll**, notando que a variável **cards** é um array de elementos, mais propriamente, um array de todas as cartas existentes e visíveis no browser.

Abaixo apresenta-se uma imagem onde é possível ver o código html como as cartas estão especificadas.

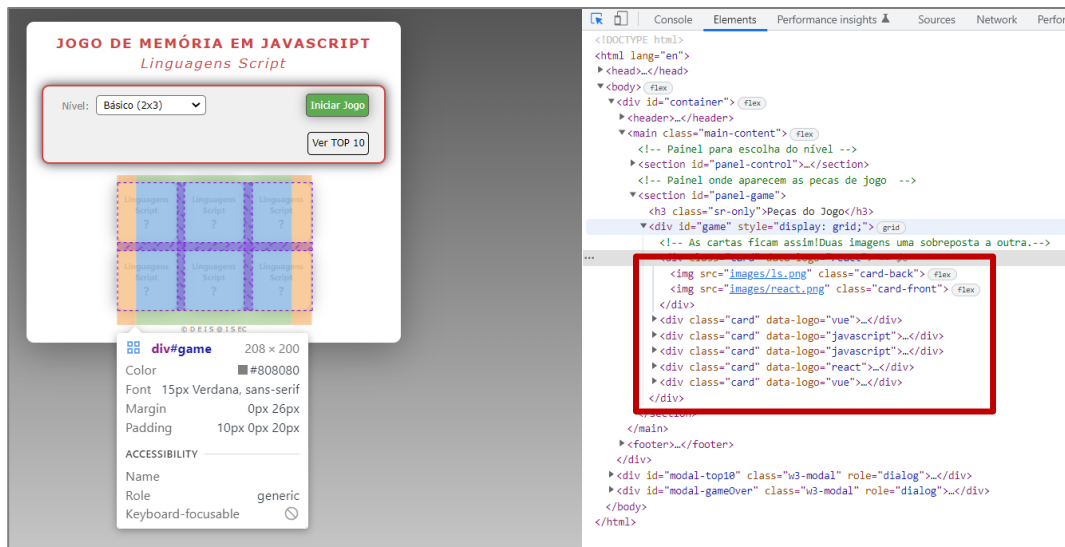


Figura 5 - Jogo de Memória – Cartas

- Implemente uma função de nome **showCards** que pretende virar todas as cartas do painel. Para isso, implemente um ciclo **for...of** que percorra o array **cards** e em cada elemento existente no array, adicione a classe **flipped**. Invoque esta função na função **startGame** já existente.
- Implemente a função **hideCards** que implementa o comportamento contrário, isto é, que remova a classe **flipped** a cada um dos elementos do array. Invoque esta função na função **stopGame**.
- Visualize o jogo no browser e certifique-se que ao carregar no botão Iniciar/Terminar jogo, as cartas efetuam uma rotação.



Figura 6 - Cartas Viradas

**g.** 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 7 - Cartas com diferentes logotipos

**h.** 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 a que o **panelGame** fique com o aspeto da figura 9, isto é, existe sempre o par de cada carta.

Algumas ideias, de entre várias formas possíveis:

- Criando um array **newCardLogos** que deve conter apenas **os 3 primeiros elementos do cardLogos**, após este estar baralhado, **e depois duplicar o array otido**. Consulte a secção dicas para fazer isso: método *splice*, ou método *slice*, método *push*, operador *spread* ...
- Especificando uma **flag Indice**– Inicie-a a flag 0 e quando atingir metade das cartas necessárias, reinicia a flag a 0 (atenção que nesta situação, será necessário baralhar cartas com *order*, por exemplo).



Figura 8 - Cartas Baralhadas

## Parte II – Action Listener para Rodar Carta

**3>** Nesta fase, pretende-se especificar o código necessário para que, ao clicar numa carta, a mesma vire, isto é, efetue uma rotação.

- a. Por forma a implementar este comportamento, coloque em comentário a invocação da função `showCards` que foi utilizada para verificar se o processo de baralhar as cartas decorria como pretendido.
- b. Especifique uma função `flipCard(selectedCard)` que a deve rodar a carta `selectedCard` recebida por parâmetro.
- c. Implemente agora o *action listener* para cada carta, invocando a função `flipCard` quando houver um clique. Assim, crie um ciclo que percorra todas as cartas (ou adicione ao ciclo já existente no `startGame`) e adicione o *action listener*. Para Invocar a função `flipCard` no *action listener*, recorra a uma função anónima, de forma a enviar argumentos para a função `flipCard`.

→ Invoque a função passando por argumento a carta, podendo recorrendo à propriedade `currentTarget`, como apresentado abaixo, ou então usando como argumento, a palavra-chave `this`

```
flipCard(e.currentTarget)
flipCard(this)
```

- d. Verifique no browser a rotação das cartas, sempre que existe um clique sobre ela.

**4>** É possível simplificar o código implementado em Parte II **3>c)** alterando a função `flipCard` de forma a não receber qualquer elemento por parâmetro, e aceda ao elemento a rodar através da **palavra chave** `this` do seguinte modo:

```
this.classList.add('flipped');
```

Para usar este método, altere ainda forma como está a invocar a função `flipCard` no *actionListener*, simplificando da seguinte forma:

```
card.addEventListener('click', flipCard);
```

## > Parte III, Parte IV e Parte V – Para explorar em casa....

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

5> Como referido anteriormente, é possível baralhar as cartas recorrendo à propriedade **order** do *grid layout* (CSS). Para isso, implemente os seguintes passos:

- Crie a variável **cards**.
- Esta variável deverá obter todos os elementos especificados com a classe **.card**, que se encontram dentro do **panelGame**. Abaixo apresenta-se imagens onde pode ver o código html e CSS de como as cartas estão a ser especificadas.

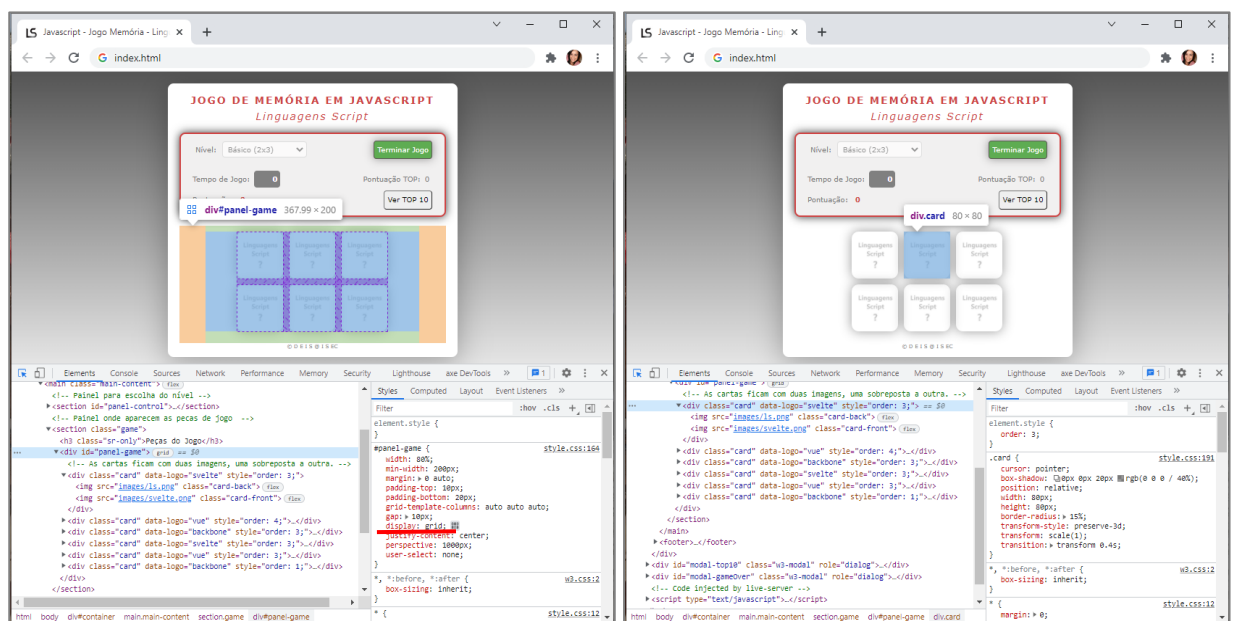


Figura 9 - Panel-game HTML + CSS

- 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 todas 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.
- De forma a verificar se cartas ficam em posições diferentes sempre que faz o *refresh* à página, implemente a função **showCards(cards)**, que deve fazer a rotação das cartas de forma a ver o logotipo da carta.
  - Para efetuar este comportamento a função **showCards** deve implementar um ciclo para percorrer todas as cartas existentes e aplicar a classe **flipped** a cada uma delas.
  - Invoque função **showCards** depois do ciclo implementado na função **startGame**.
- Faça vários *refresh* à página e certifique-se que obtém o comportamento desejado, isto é, que as cartas se posicionam em diferentes posições.



Figura 10 - Cartas aleatórias e viradas

### Parte III – Alterar o contorno por Javascript

- 6> 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)

- 7> 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.