

> 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 e rodar cartas. Tal como na ficha anterior. O HTML, bem como o CSS necessário à resolução, já inclui todos os elementos necessários.
- Recomenda-se a consulta das dicas, durante a resolução, apresentadas na seção **Dicas**.
- O resultado final da ficha apresenta-se na figura seguintes.



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:

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

- Para alterar elementos de um *array*:

```
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 manipulação. 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 = nomeArray.slice(0, 4); // Devolve elementos entre índice 0 e 4
nomeArray.splice(1, 1, 'novo1', 'novo2'); // Inserir elementos dentro do array
```

- a.** 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/alterar os dados de um atributo **data**, pode recorrer à propriedade **dataset** como se apresenta no exemplo seguinte:

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

- b.** Quando se adiciona um *Event Listener* com `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) {...}
```

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

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

d. 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

> 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**, de forma a que as cartas fiquem com o aspeto da figura 2.

- 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.



Figura 2 - Jogo (início)

> Explicação de algum código HTML e regras CSS

- a. O comportamento existente de especificar um contorno quando o rato passa em cima de uma carta, está implementado sem recorrer a qualquer JavaScript. Apenas foram usadas regras CSS, que pode encontrar na folhas de estilos cujo ficheiro é **index.css** (selector **.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 encontrado no ficheiro **index.html**. Como pode verificar, uma carta é composta por duas imagens, a imagem **ls.png** e a imagem **react.png**.

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

As classes **card-back** e **card-front** contêm regras CSS de forma a que as duas imagens fiquem sobrepostas e, além disso, a imagem com classe **card-front** inclui uma rotação 180° no eixo dos Y de forma a permitir o efeito de rotação da carta, quando depois houver um clique.

Parte I – Baralhar as Cartas

1> Nesta fase, pretende-se especificar o código necessário para baralhar as cartas existentes no painel de jogo. 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.

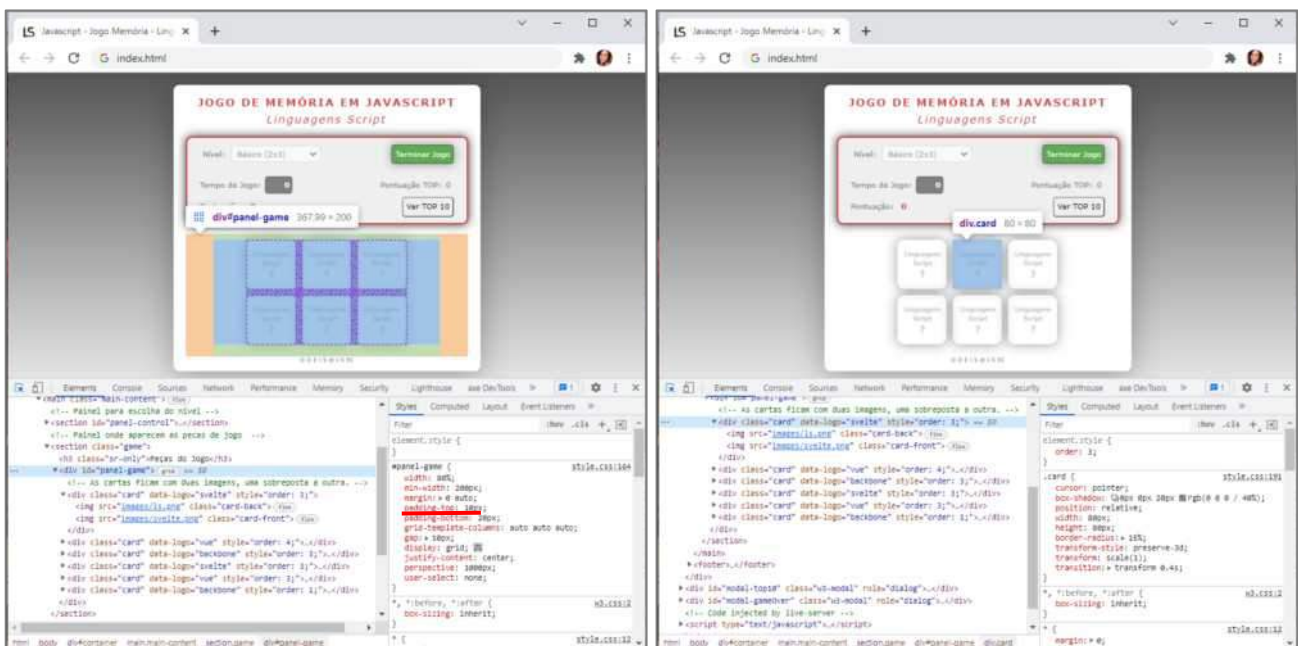


Figura 4 - Panel-game HTML + CSS

- 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 como 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.

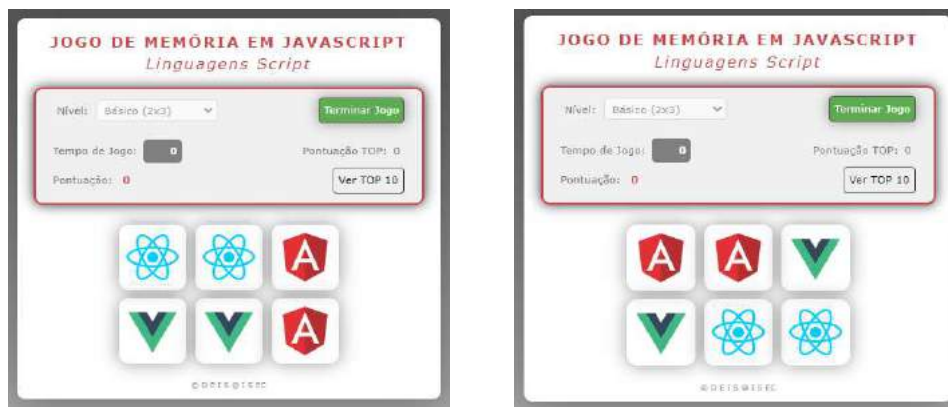


Figura 5 - Cartas aleatórias e viradas

2> Como pode verificar na figura 6, os logotipos são os mesmos, apenas em posições diferentes. Assim, pretende-se que para além da ordem ser diferente, que os logotipos também sejam diferentes, de forma aleatória.

a. Para alterar os logotipos, implemente os seguintes passos:

- Declare o array **cardsLogos** com os seguintes valores:
 - angular
 - bootstrap
 - html
 - javascript
 - vue
 - svelte
 - react
 - css
 - backbone
 - ember

- 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;
  }
}
```

- Ainda dentro da função **startGame**, continuando o código anteriormente implementado, invoque a função **shuffleArray(cardsLogos)**.
- Para verificar o estado do *array*, imprima na consola o *array* antes e depois da chamada à função de forma a ver se, de facto, o *array* **cardsLogos** passou os seus *items* baralhados, como exemplo, se apresenta na figura 6.

console.table(cardsLogos)

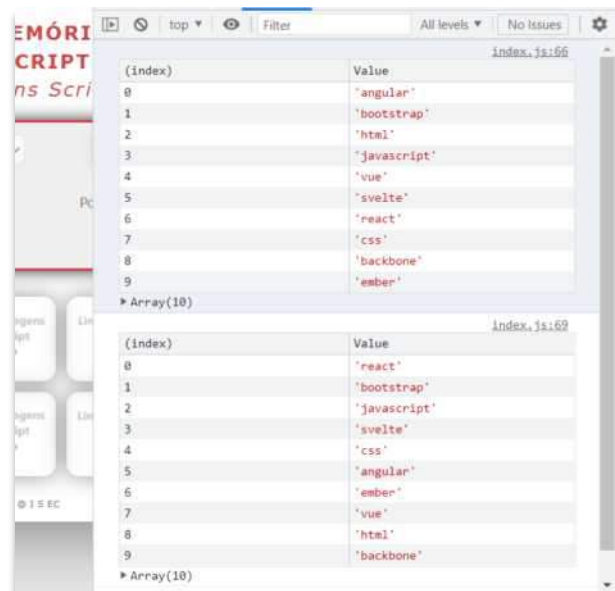


Figura 6 - Apresentação do array na consola

- b.** Existem várias imagens na pasta **images**, em que o nome do ficheiro, é igual ao nome dos items do array **cardsLogos**, e portanto, será esse o ponto de ligação entre o nome ficheiro e o array.

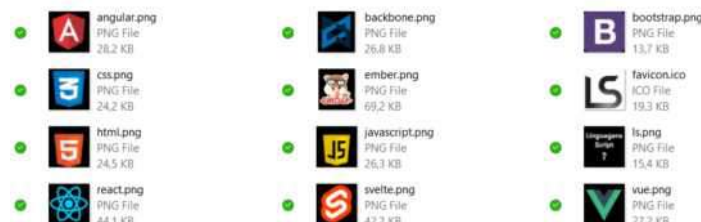


Figura 7 – Imagens

- c.** Com o array **cardsLogos** baralhado, pretende-se alterar efetivamente as cartas apresentadas no browser. Nesse sentido, e verificando o trecho de código HTML apresentado abaixo, com especial atenção aos que estão destacados, verifique que:

- Se a carta apresenta o logotipo javascript, o atributo data-logo é Javascript, bem como o nome do ficheiro, src é javascript.png.:

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



- Com recurso a um ciclo existente em `startGame`, percorra todas as cartas existentes, alterando os dados da carta de acordo com os itens do array `cardsLogos` baralhado. Por exemplo, se o array estiver com os valores `['react', 'backbone', 'ember', 'svelte', 'css', ...]`, a primeira carta deverá ficar:

```
<div class="card" data-logo="react">
  
  
</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. Deve alterar recorrendo à propriedade `src` do carta.

- d. Visualize no browser o comportamento obtido.** Deverá ter obtido situações como na figura seguinte, figura nº 8. **Como pode reparar, as cartas têm diferentes logotipos e não é esse o objetivo do jogo de memória, no qual devem existir pares de logotipos.**



Figura 8 - Cartas com diferentes logotipos

- e. Assim, 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:

- **Uma forma:** Especificar uma *flag* de forma a que inicie a 0 quando atingir metade das cartas necessárias,
- **Outra forma:** Crie o array **newCardLogos** que deve conter apenas os 3 primeiros elementos do **cardLogos**, após estar baralhado, e depois duplique o array otido. Consulte a secção dicas para fazer isso: método *slice* e o operador *spread* ...



Figura 9 - 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 rode.
- a.** Por forma a implementar o desejado, coloque em comentário a função `showCards(cards)` que foi utilizada para verificar se o processo de baralhar as cartas decorria como pretendido.
 - b.** Especifique uma função `flipCard(c)` que a deve rodar a carta `c` recebida por parâmetro, utilizando o mesmo raciocínio para rodar a carta na função `showCards`.
 - 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 `addListener`, simplificando da seguinte forma:

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

> Parte III e Parte IV - Se houver tempo ou para explorar mais tarde

Parte III – Alterar o contorno da carta efetuado por CSS 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 foram usadas regras CSS, que pode encontrar no ficheiro *index.css*. Pretende-se nesta secção, efetuar o mesmo comportamento, mas agora recorrendo ao *JavaScript*. Assim, implemente os seguintes passos:



Figura 10 - 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 listener* 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 mantem.

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.