> *Ficha Prática Nº4 (Jogo de Memória – Lógica do Jogo)*

Notas:

- **Esta ficha, tem como objetivo implementar as funções necessárias para concluir a lógica do jogo de memória, identificando as cartas pares e voltando as cartas caso não formem pares. Pretende-se ainda identificar fim de jogo quando todas os pares forem identificados.**
- Os alunos n**ão devem alterar** o documento HTML nem os ficheiros de estilos existentes, de forma a seguirem o propósito da ficha, **nem 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*.
- O resultado final da ficha apresenta-se na figura 1.

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

> *Preparação do ambiente*

a. Efetue o download e descompacte o ficheiro **ficha4.zip** disponível no *inforestudante*.

NOTA: Os alunos que concluíram a resolução da ficha 3, podem usar essa versão.

Por curiosidade, podem analisar o JavaScript fornecido nesta ficha.

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 (inicio)

Parte I – Verificar se formam Pares

1> Nesta fase, pretende-se especificar o código necessário para verificar se, após rodar duas cartas, as mesmas são pares ou não. Para isso implemente os seguintes passos:

a. A função `flipCard` realizada na ficha 3 (fornecida também como base nesta ficha), implementa o código que permite virar a carta recebida por parâmetro, aplicando classe `'flipped'`.

```
function flipCard(selectedCard) {
    selectedCard.classList.add('flipped');
}
```

b. Por forma a verificar se duas cartas são pares, declare o array `flippedCards` e inicialize-o sempre que um novo jogo começa, portanto, na função `startGame`. Este array `flippedCards` irá armazenar sempre as duas cartas viradas (ver passo c).

c. Na função `flipCard`, adicione código de forma a que a carta recebida por parâmetro seja adicionada ao *array* `flippedCards`, usando por exemplo o método **push** que permite adicionar novos elementos a um *array*.

Ainda na função flipCard, verifique se já existem dois elementos no *array* `flippedCards` e, em caso afirmativo, invoque a função `checkPair()` que se irá implementar de seguida e que vai verificar se as duas cartas formam ou não um par.

d. Como referido na alínea anterior, a função `checkPair()` deve verificar se as cartas foram um par. Para isso, deverá comparar o atributo `data-logo` de cada uma das cartas, isto é de cada um dos elementos *card* existentes no array `flippedCards`. Assim, faça com que a função escreva "**Iguais**" na consola, se forem iguais, caso contrario, deve escrever "**Não são iguais**". Além disso, e como são duas cartas já viradas, independentemente de serem ou não pares, faça um *reset* ao *array* `flippedCards` de forma a eliminar as cartas existentes no *array*.

Note que, para comparar as duas cartas, basta comparar o atributo data-logo. Caso os dois elementos tenham o mesmo texto, são então iguais.

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

e. Confirme no **browser** e na consola, se está a identificar corretamente se as cartas formam ou não um par.

2> Nesta secção, pretende-se **implementar o código quando duas cartas formam um par**, isto é, quando as duas cartas são iguais!

a. Na função `checkPair()` implementada anteriormente, adicione, a cada uma das duas cartas existentes no *array* **flippedcard**, a classe `inative` (que retira o contorno existente), **quando as duas cartas são iguais**.
Além disso, aplique ao elemento com classe `card-front`, de cada uma das cartas, a classe `grayscale` que faz com que a imagem passe a ficar na escala de cores cinza. A figura 3 apresenta o aspeto quando duas cartas iguais foram rodadas.

Figura 3 - Cartas Pares (Iguais)

b. Confirme o código implementado no **browser** e na consola. Tente encontrar um par igual.

3> Nesta secção, pretende-se **implementar o código quando as duas cartas não formam par.**

a. Na função `checkPair()`, quando as duas cartas não são iguais, remova a cada uma dessas cartas, a classe `flipped` que permite voltar a carta.

b. Confirme o código implementado no **browser** e na consola.

c. Como pode verificar, quando as cartas não são iguais, não é possível ver a rotação da segunda carta, uma vez que o comportamento de voltar às posições originais é efetuado rapidamente. Assim, deverá recorrer ao método `setTimeout` que permite especificar um temporizador para executar uma determinada função ou bloco de código quando o tempo definido terminar (timeout). Repare que o `setTimeout` é uma função assíncrona, o que quer dizer que a função do tempo não irá parar a execução de outras funções. Assim, **todo o bloco de código existente** na secção *"Não são iguais"* deve ser incluído dentro deste método.

```
setTimeout(() => {
        ... //Codigo
}, 500);
```

d. Confirme o código implementado **browser**. Neste momento, já deve ter o comportamento desejado, como se apresenta na figura seguinte:

e. Se pretender, pode também incluir o método `setTimeout` quando as cartas são iguais, de forma a alterar o aspeto apenas depois de alguns milissegundos.

Figura 4 - Identificação de pares ou não

4> A resolução anterior, apresenta um problema que se pretende resolver neste momento.

Figura 5 - Clicar 2 vezes na mesma carta.

a. Verifique no browser o que acontece se clicar duas vezes na mesma carta. Como pode verificar, como apresentada na figura 5, é considerada carta igual.

b. Uma das formas de resolver este problema é alterar a forma como está especificado o *Event Listener* que reage ao clique na carta, fazendo com que reaja ao evento uma única vez ***once:true***. Além disso, e de forma a poder ser possível remover o evento associado ao click, em vez de usar uma função anónima, especifique o nome a função. Para isso, **altere o código já existente** relativamente ao **addEventListener**, da seguinte forma, o qual quando uma carta for clicada, já na pode ser novamente clicada, a não ser que se adicione novamente com `addEventListener.`

Portanto, substituir o seguinte código:

```
card.addEventListener('click', function () {
    flipCard(this);
});
```

Pelo seguinte:

```
function funcClickCard(e) {
    flipCard(this);
}
...
card.addEventListener('click', funcClickCard, { once: true });
...
```

c. Verifique no browser que o comportamento ficou corrigido, no entanto, quando as cartas são diferentes e voltam à posição anterior, não é possível voltarem a virar.

Assim, quando as cartas são diferentes na função `checkPair()`, onde está a remover a classe **flipped** para voltarem à posição original, adicione novamente o *Eventlistener*, como anteriormente apresentado, para estas duas cartas.

d. **Verifique no browser o comportamento**, o qual deverá ficar com o problema resolvido.

Parte II – Fim de Jogo

5> Identificação de fim de jogo.

a. Uma forma de identificar fim do jogo, é verificar se todas as cartas foram viradas. Assim, declare a variável a **totalFlippedCards**, e esta deverá ser inicializada a 0 sempre que se inicia um jogo, portanto, na função `startGame`.

b. Crie a função `gameOver()` que deverá devolver `true`, caso o jogo tenha atingido o fim (numero de cartas viradas = numero total de cartas) ou `false` caso contrario.

c. Sempre que as cartas forem iguais, incremente a variável **totalFlippedCards** em 2 unidades e verifique se é fim de jogo invocando a função `gameOver()`. Caso seja fim de jogo, deverá invocar a função `stopGame()`

d. Na função `stopGame()` apresente a janela modal de fim de jogo com a seguinte linha de código **modalGameOver.style.display = 'block';.**

e. Por fim, na função `stopGame()`, **remova** todas as classes aplicadas a cada **card** existente em **cards**, nomeadamente:

- → `flipped,`
- → `inative`
- → `grayscale`

Desse modo, quando se voltar a iniciar um jogo, as cartas estarão com o comportamento correto.

Além disso, ainda na função stopGame, deverá remover o *EventListener* associado ao click através do método **removeEventListener('click', nomeDaFuncao);** para que as cartas não rodem quando o jogo já tiver parado.