> *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 os pares e voltando as quando estas não são pares, identificando ainda fim do jogo.
- 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 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` que deverá armazenar duas cartas viradas. Este *array* deverá ser inicializado sempre que um novo jogo se inicia, portanto, na função `startGame`.

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*.

Nessa mesma função, verifique se já existem dois elementos no *array*, e em caso afirmativo, invoque a função `checkPair()`, a implementar, função esta que deve verificar se as cartas são pares ou não. Para efetuar esse passo, dentro dessa função, deverá comparar o atributo `data-logo` de cada uma das cartas. Se forem iguais, escreva na consola "**Iguais**", caso contrario deverá escrever "**Não são iguais**". Além disso, faça um reset ao *array* `flippedCards` de forma a eliminar as cartas existentes no array.

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

d. Confirme no **browser** e na consola, se está a identificar corretamente se as cartas são ou não pares.

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

a. Na função `checkPair()`, quando as duas cartas são iguais adicione a cada uma dessas cartas, a classe `inative` que permite retirar o contorno existente. Além disso, aplique ao elemento com classe `card-front`, de cada uma das cartas, a classe `grayscale` para colocar a imagem com escala de cores cinza. A figura 3 apresenta o aspeto quando duas cartas iguais foram rodadas.

b. Confirme o código implementado **browser** e na consola.

Figura 3 - Cartas Pares (Iguais)

3> Nesta secção, pretende-se **implementar o código quando as duas cartas Não são pares**!

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 **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 terminar. Desse modo, todo o bloco de código existente na secção *"Não são iguais"* deve ser incluída dentro deste método.

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

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

Figura 4 - Identificação de pares ou não

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

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

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.

Figura 5 - Clicar 2 vezes na mesma carta.

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. Para isso, altere o código existente 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.`

```
card.addEventListener('click', function () {
    flipCard(this);
}, { 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 e remove a classe flipped para voltarem à posição original, adicione novamente o eventlistner, como anteriormente apresentado, mas apenas 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`, que deverá ser inicializada a 0 na função `startGame`, e incremente a variável sempre que forem identificadas cartas pares.

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, e aumentar o numero de cartas viradas, verifique se é fim de jogo e 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 além disso remova todas as classes aplicadas às cartas/imagem, nomeadamente: `flipped, inative, grayscale.`

Parte III – Pontuação (versão 1)

6> Pretende-se especificar pontuação em cada jogada.

a. Numa primeira fase pontue o jogo recorrendo às seguintes regras:
   - Para uma jogadada correcta, incremente à pontuação existente o total de cartas que falta identificar * 2.
   - Para uma jogada incorrecta, deverá ser decrementado 2 pontos, ou, caso não tenha 2 pontos, deverá passar a ter a pontuação 0.

b. Implemente o código de forma a atualizar a pontuação no painel, na zona de pontuação.