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

Esta ficha tem como objetivo implementar as funções necessárias para concluir a lógica do jogo de memória. Assim, pretende-se **identificar pares de cartas iguais existentes no tabuleiro e, nessa situação, as cartas fiquem definitivamente voltadas, caso contrário, as cartas devem voltar à posição original**. Pretende-se ainda **identificar fim de jogo** quando todas os pares de cartas forem identificados.

O resultado da ficha apresenta-se na figura seguinte:

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

> *Preparação do ambiente*

a. Descompacte o ficheiro **ficha4.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 juntamente com esta ficha.

b. Inicie o *Visual Studio Code* e abra **a pasta no *workspace*.** Visualize a página **index.html** no browser, no qual terá o aspeto da figura 2.

Figura 2 - Jogo (inicio)

Parte I – Verificação de Pares

1> Nesta fase, pretende-se especificar o código necessário para verificar se, após rodar duas cartas existentes no tabuleiro de jogo, as mesmas são pares ou não. Para isso implemente as seguintes alíneas.

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

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

b. Por forma a verificar se duas cartas são pares, declare o array **`flippedCards`** no *scope global* e inicialize-o sempre que um novo jogo começa, portanto, na função `startGame`. Este array **`flippedCards`** tem como objectivo armazenar as duas cartas viradas (ver passo c).

c. Na função `flipCard`, adicione código de forma a que a carta virada (obtida através da palavra-chave *this*) seja adicionada ao *array* **`flippedCards`**, usando, por exemplo o método **push** que permite adicionar novos elementos a um *array*.

Ainda nesta função flipCard, verifique se já existem dois elementos no *array* **`flippedCards`** e, **em caso afirmativo**, invoque a função `checkPair()` que se implementa de seguida e no qual vai verificar se as duas cartas existente no array **`flippedCards`** formam ou não um par (isto é, se tem o logotipo igual).

d. Como referido na alínea anterior, a função `checkPair()`tem objetivo **de verificar se as cartas foram um par (se são iguais)**. 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, numa primeira fase, faça com que a função `checkPair:`

- escreva "**Iguais**" na consola, se as duas cartas existentes no array forem iguais, caso contrario, deve escrever "**Não são iguais**". Note que, para comparar as duas cartas, basta comparar o atributo **data-logo**. Por exemplo: se as duas cartas tiverem texto `javascript` no **data-logo** é porque são iguais.

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

- Uma vez que estão duas cartas já viradas, independentemente de serem ou não pares, faça um *reset* ao *array* **`flippedCards`** de forma a eliminar as cartas existentes nesse array.

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 para especificar o comportamento seguindo quando duas cartas formam um par**, isto é, quando as duas cartas são iguais. Assim, quando as cartas viradas são iguais, as cartas devem ficar **com a cor em escala cinza (**aplicar a class `grayscale` elemento com class **card-front**), e ficarem desativadas (sem qualquer comportamento aquando de um clic*k*), até que o jogo termine. Para isso, implemente os seguintes passos.

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.** Nesta situação, seguindo a lógica de um jogo de memória, as cartas voltam novamente à posição original, isto é, rodam até o par ser encontrado. Assim, implemente os seguintes passos.

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 à posição inicial.

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 código de voltar às posições originais é efetuado tão rápido, que não se vê o comportamento de rotação da carta. Assim, é necessário é necessário "atrasar" o processo para se ver a carta a voltar à posição anterior.

Para isso, 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, **o bloco de código existente** na secção *"Não são iguais"* deve ser incluído dentro da invocação deste método.

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

d. Confirme o código implementado **browser**. Neste momento, já deve ter o comportamento desejado, como se apresenta na figura 4.

Figura 4 – Identificação ou não de pares

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.

4> Talvez não se tenha apercebido, mas, 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 efetuar dois cliques na mesma carta.** Como pode verificar, como se apresenta na figura 5, **é considerada carta igual.**

Ao analisar o seu código, deve chegar à conclusão que esse comportamento acontece porque, por cada clique que se efetua na carta, adicionamos a carta no array **flippedCards** e, então, a mesma carta é adicionada duas vezes (correspondente aos dois cliques). Como de seguira irá executar o **checkpair**, as cartas são identificadas, erradamente, como iguais. Portanto, para resolver esta situação, é necessário aceitar **apenas um clique numa carta** e desativar o seu comportamento até que outra carta do tabuleiro seja clicada ou então, a mesma, seja voltada à posição original.

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***. Para isso, **altere o código já existente** relativamente ao **addEventListener**, da seguinte forma, o qual quando uma carta for clicada, já não pode ser novamente clicada, a não ser que se adicione novamente com `addEventListener`. Portanto, deve substituir o código `card.addEventListener('click', flipCard);` pelo seguinte:

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

Verifique no browser que o comportamento ficou corrigido, no entanto, repare que, quando as cartas são diferentes e voltam à posição anterior, não é possível voltarem a virar, porque o *event listener* foi "desativado". Para resolver isso, quando as cartas são identificadas como diferentes na função `checkPair()`, onde está a remover a classe **flipped** para voltarem à posição original, adicione novamente o *Eventlistener*, como anteriormente apresentado, para as duas cartas.

Alternativa: *Uma outra forma de resolver esta situação, sem recorrer ao código* ***once:true****, pode ser através da remoção do eventlistener associado ao click, usando o método* ***removeEventListener****.*

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

Parte II – Fim de Jogo

5> Identificação de fim de jogo. Uma das formas de fim do jogo, é verificar se todas as cartas já foram viradas, tendo em consideração o número total de cartas. Assim, implemente os seguintes passos.

a. Declare a variável a **`totalFlippedCards`**, *no scope global,* e inicialize-a 0 sempre que se inicia um novo jogo, portanto, na função **`startGame`**.

b. Crie a função **`gameOver()`** que deverá devolver `true`, caso o jogo tenha atingido o fim (nº de cartas viradas = nº 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.showModal();
```

Nota: Se na função **stopGame** estiver a invocar a função *hideCards*, realizada na última ficha, coloque-a em comentário, pois não será mais necessária.

e. Ainda, na função **`reset()`**, **remova** todas as classes aplicadas a cada **card** existente em **cards**. Assim, implemente um ciclo, por exemplo com o *`foreach ou for…of`*, e remova as seguintes classes que se encontram aplicadas a cada card:

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

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

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