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

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 no tabuleiro** e, caso sejam encontrados, mantê-las **viradas permanentemente**; caso contrário, as cartas devem voltar à **posição original**. Além disso, pretende-se **apresentar o tempo de jogo** e, quando todos os pares forem identificados ou o tempo se esgotar, o **fim de jogo** deve ser exibido. As imagens seguintes, apresentam o resultado pretendido:

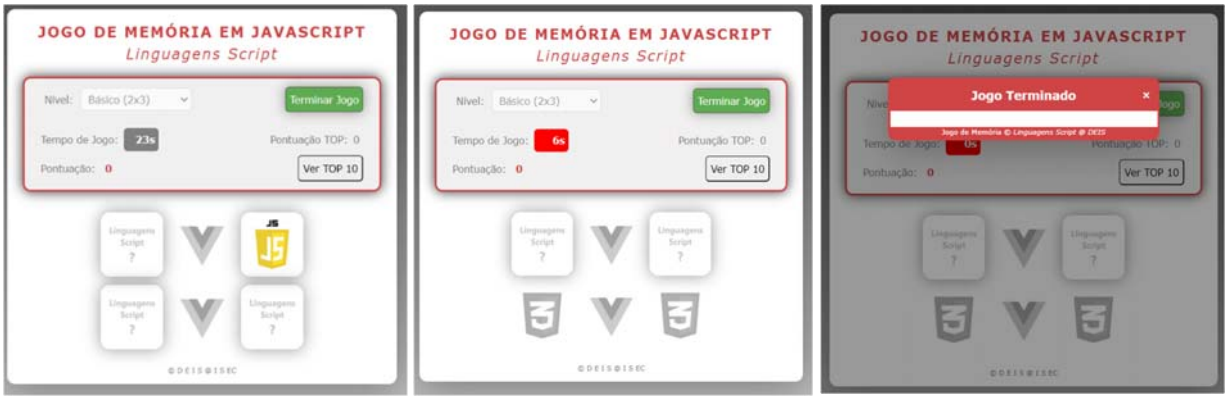

Figura 1 – Jogo de Memória em JavaScript – Ficha Prática Nº4

> *Preparação do ambiente*

a. Descompacte o ficheiro **ficha4.zip**.

Os alunos que concluíram a resolução da ficha anterior, devem continuar nesse projeto. 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.

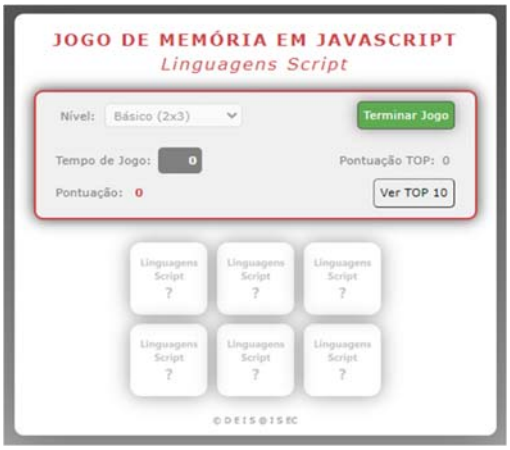

Figura 2 - Jogo (início)

c. *Módulos a implementar:*

- → **Verificação de Pares (Parte I)**, que se altera em cada jogada, aumentando quando é encontrado um par de cartas ou diminuindo quando jogador falha o par.
- → **Fim de Jogo (Parte II)**, onde se identifica o fim de jogo quando é interrompido ou porque todas as cartas foram viradas.
- → **Tempo de Jogo (Parte II)**, onde é iniciado com valores diferentes de acordo com o nível selecionado, e é decrementado durante o jogo.

Parte I – Identificação de Pares

1> Nesta fase, pretende-se especificar o código para identificar se, após virar duas cartas do tabuleiro, formam um par ou não. Para isso implemente as seguintes alíneas.

a. A função `flipCard` realizada na ficha anterior (fornecida também como base nesta ficha), contém o código que permite virar a carta quando 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`. O objetivo deste array **`flippedCards`** é armazenar apenas as duas cartas viradas (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 par (isto é, se o logotipo igual, mais precisamente se o atributo data-logo é igual).

d. Como referido na alínea anterior, a função `checkPair()`tem objetivo **de verificar se as cartas formam 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:`

1) escreva "**Iguais**" na consola, se as duas cartas existentes no array forem iguais; caso contrário, escreva "**Não são iguais**". Note que, para comparação das duas cartas, apenas precisa comparar o atributo **data-logo**. Por exemplo: se o atributo no **data-logo** das duas cartas for `javascript` é porque são iguais.

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

2) 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 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 permanentemente nessa posição e **com a cor em escala cinza (**aplicar a class `grayscale` elemento com class **card-front**), e ficarem desativadas (sem qualquer comportamento aquando de um *click*), 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* **flippedCards**, a classe `inactive` (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.

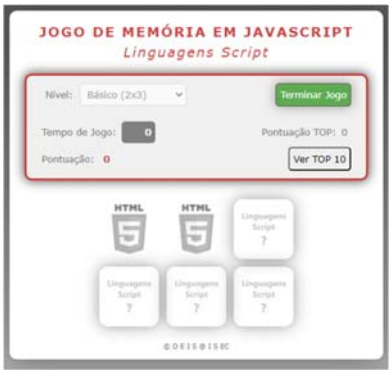

Figura 3 - Cartas Pares (Iguais)

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

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

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 ou não de pares

4> Talvez não se tenha apercebido, mas, a resolução anterior, apresenta um problema que se pretende resolver neste momento.

a. **Verifique o que acontece ao efetuar dois cliques na mesma carta.** Como deve ter reparado, e como se apresenta na figura 5, **os dois cliques na carta são considerados, e é identificado, erradamente, como duas cartas iguais.**

b. Esse comportamento acontece, pois, por cada clique que se efetua na carta, esta é adicionada no *array* **flippedCards** e, então, a mesma carta é adicionada duas vezes (correspondente aos dois cliques).

Figura 5 - Clicar 2 vezes na mesma carta.

Como posteriormente é executada a função **checkPair**, as cartas são identificadas, erradamente, iguais. Portanto, para resolver esta situação, é necessário aceitar **apenas um clique na carta** e desativar qualquer comportamento nela até que volte à posição anterior ou o jogo termine.

Existem várias formas de resolver esta situação, entre elas:

1) Verificar, no início da função `flipCard()`, se a carta clicada já tem a classe *flipped* aplicada, antes da própria função a aplicar, indicando assim que a carta já se encontra virada, e, nessa situação, interromper a execução da função flipCard. Para implementar este comportamento, bastaria adicionar `if (this.classList.contains("flipped")) return;` no início dessa função, e o problema fica resolvido;

2) Remover o *eventlistener* associado à carta sempre que esta é virada, recorrendo ao método *removeEventListener*. Note que será necessário adicionar novamente o *listener*, caso a carta volte a posição original;

3) Por fim, outra forma de resolver a situação, será alterar a forma como o *addEventListener* especificado, fazendo com que reaja ao evento uma única vez. Para isso, o código `card.addEventListener('click', flipCard);` deverá ser substituído por:

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

Com este parâmetro, a carta invoca a função flipCard quando for clicada, uma única vez, a não ser que se adicione novo *event listener.*

Note que, quando as cartas são diferentes e voltam à posição original, não é possível voltarem a efetuar qualquer rotação, pois o *event listener* foi removido. 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 e*ventListener*, como anteriormente apresentado, para as duas cartas.

c. **Verifique no browser o comportamento do jogo**, que deverá ter 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 de expressão `gameOver` que devolva `true`, caso o jogo tenha atingido o fim (nº de cartas viradas = nº total de cartas) ou `false` caso contrário.

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()`. Quando for fim de jogo, deverá ser invocada a função `stopGame()`

d. Na função `stopGame()` apresente a janela modal de fim de jogo com a seguinte linha de código, e remova a invocação à função reset (uma vez que será invocada quando a modal for fechada).

```
//reset();
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. Por fim, na função `reset()`, **remova** as classes `flipped` e `inactive` aplicadas a cada **card** existente no array **cards**, e a classe `grayscale` que se encontra aplicada à imagem **card-front**. Implemente com recurso a ciclo *foreach ou for…of*.

f. **Verifique o jogo no browser.**

Parte III – Tempo de Jogo

Nesta secção, pretende-se especificar o código necessário para limitar o tempo de jogo, no qual o tempo inicia com determinado valor, de acordo com o nível, e decrementa até chegar aos 0s. **Nos últimos 10 segundos**, a cor de fundo onde o tempo é apresentado deverá ser destacado a vermelho, como se apresenta nas figuras seguintes.

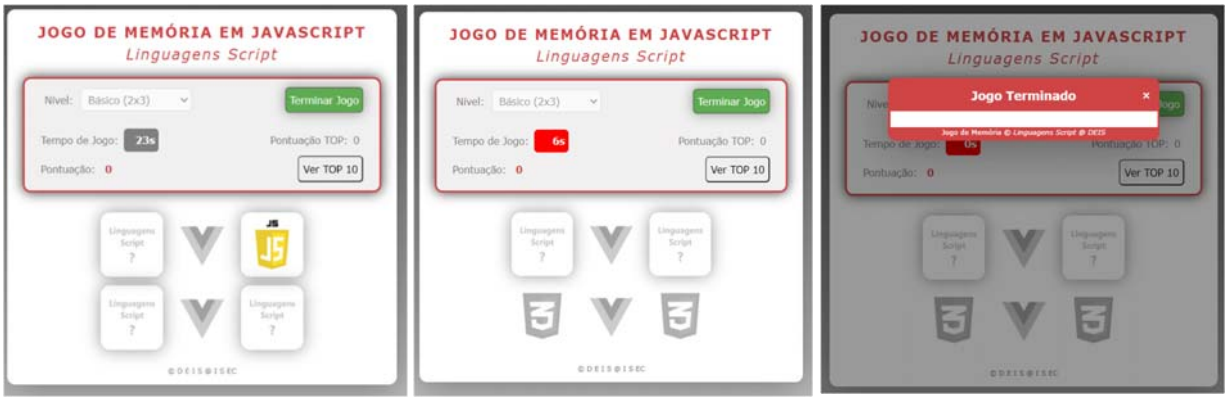

Figura 6 - Tempo de Jogo

6> Para especificar o tempo de jogo será necessário recorrer ao método `setInterval`. É uma função assíncrona, o que quer dizer que a função do tempo não irá parar a execução de outras funções, e permite **executar código ou invocar uma função repetidamente**, com um t**empo de espera fixo entre cada execução. No caso do jogo, pretende-se alterar que o tempo de jogo apresentado ao jogador, seja alterado a cada segundo.** O `setInterval` pode ser cancelado recorrendo ao `clearInterval()`.

a. Para efetuar o comportamento desejado, declare as seguintes variáveis:

- A constante `TIMEOUTGAME`=20 (considerando que o jogo irá durar 20s).
- `labelGameTime` que deve aceder ao elemento da página cujo id é `gameTime`, que será necessária para poder atualizar o tempo de jogo na aplicação.
- **`timer`** que irá armazenar o tempo de jogo e irá ser alterado durante o jogo.
- **`timerId`** para ser possível interromper a execução da função `setInterval` (a ver de seguida);

b. Na função `startGame`:

- Inicialize a variável **`timer`** com o valor da constante `TIMEOUTGAME`
- Apresente o texto "**20s**" com recurso à propriedade textContent do `labelGameTime`
- Especifique o `timerId` da seguinte forma

```
timerId = setInterval(updateGameTime, 1000)
```

Note que a função `setInterval` irá executar a função `updateGameTime` (a implementar de seguida), a cada segundo (1000 = 1 segundo). Só irá parar, quando se cancelar este comportamento com a função `clearInterval`.

c. Implemente a função **updateGameTime** que deverá:
   - Decrementar a variável **timer**
   - Atualizar o tempo em labelGameTime recorrendo à propriedade textContent

d. Por fim, na função **stopGame**, cancele o temporizador, adicionando a seguinte linha de código, para que o tempo de jogo não seja novamente atualizado aquando término do jogo.

```
clearInterval(timerId);
```

e. Verifique **no browser o comportamento do jogo**.

f. Como pode verificar, depois de atingir o tempo de jogo 0, e o jogo ainda não tiver sido concluído, a função **updateGameTime** continua a decrementar, como se mostra na figura seguinte.

Assim, para que isso não aconteça, é necessário invocar a função **stopGame** quando o tempo de jogo chegar a 0. Essa linha de código deverá ser então aficionada na função **updateGameTime**.

g. Verifique **no browser o comportamento do jogo**, que deverá ser o desejado.

7> Altere a cor de fundo do elemento **labelGameTime**, para vermelho, quando o timer for inferior a 10.

a. Altere a propriedade *style*, para vermelho no background, na função **updateGameTime**.

b. Na função reset, a cor deverá voltar ao estado normal, isto é, basta remover o atributo *style* ao elemento, por exemplo, com recurso ao método removeAttribute

```
labelGameTime.removeAttribute('style');
```

c. **Confirme no browser o comportamento do jogo.**

8> Implemente as alterações necessárias para que o tempo de jogo, seja diferente, em cada nível. Para isso:

a. Implemente a função getTime() que **retorne** o tempo de jogo, isto é, o valor com que a variável **timer** deve ser inicializada, de acordo com o nível de jogo selecionado. Declare as seguintes constantes, no *global scope,* de forma a poderem ser usadas na função a implementar:
   - TIMEOUTGAME_BASICO = 20 // O tempo de jogo é de 20 segundos
   - TIMEOUTGAME_INTERMEDIO = 60
   - TIMEOUTGAME_AVANCADO = 180

b. Altere a inicialização da variável timer, na função **startGame**, invocando a função getTime()

c. **Confirme no browser** o comportamento do jogo, o qual o temporizador já deverá estar a comportar-se corretamente e inicializado com valores diferentes, de acordo com o nível selecionado.