> *Ficha Prática Nº2 - Jogo de Memória em JavaScript*

Pretende-se, nas próximas aulas de *Linguagens Script*, implementar, em **JavaScript**, o tradicional **jogo de memória**, que terá o aspeto apresentado das figuras seguintes.

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

O jogo tem disponível vários **níveis** (básico, intermédio ou avançado), que irão afetar o número de cartas a ser apresentadas no tabuleiro. Além disso, o tempo de jogo e uma jogada acertada ou falhada, implica alterações na pontuação final. Por fim, será possível escrever o nome do jogador, quando este entrar no top 10 dos jogadores (melhor pontuação).

O **código** relativo ao **HTML e o CSS**, já se encontra implementado e os alunos não devem, numa fase inicial, efetuar alterações no mesmo, de forma a seguirem o propósito das fichas, cujo objetivo é a criação de um jogo de memória, para aprendizagem de conceitos de programação em JavaScript. Em todo o caso, os alunos devem **analisar o código fornecido**, tanto o HTML como as classes existentes no ficheiro *styles.css*, para resolução do jogo de memória pretendido. Além disso, os alunos **não devem** remover a instrução `'use strict'` que se encontra no topo do ficheiro *index.js* de forma a ser usada, na implementação, uma variante mais restritiva do JavaScript.

Durante a realização das várias fichas, aceda ao ***Developer Tools*** do browser (ex. clicando F12 no Chrome) e selecione a tab "Console" onde poderá ver os erros de *JavaScript* da página ou simplesmente utilizar essa área como um editor de *JavaScript*. Além disso, sugere-se a instalação da extensão *Live Server* no *Visual Studio Code.*

> Preparação do ambiente – Efetue os seguintes passos:

a. **Descompacte** o ficheiro **ficha2.zip**.

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

c. O **código** relativo ao **HTML e o CSS**, já se encontra implementado.

d. A resolução desta ficha pretende:

- → **Esconder o tabuleiro de jogo, ficando visível apenas quando se seleciona um nível;**
- → **Ativar/desativar elementos do painel de controlo,** recorrendo a *event listeners.*

Figura 2 - Jogo (início)

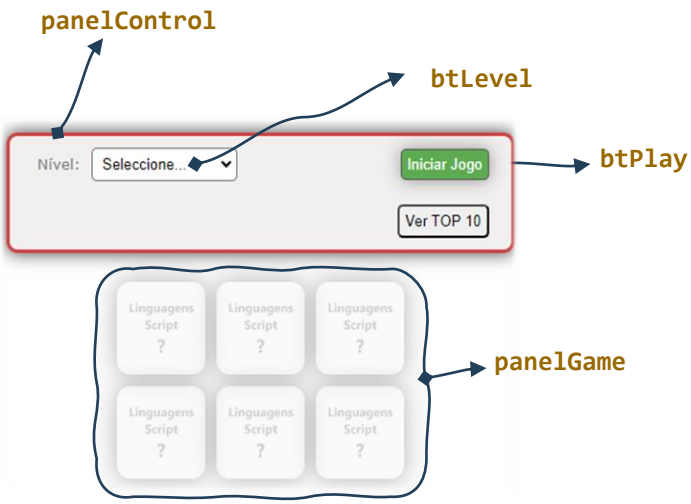
Parte I – Implementação função Reset

1> Pretende-se implementar o aspeto inicial do jogo, como apresentado na Figura 3. Para facilitar o processo, implemente as alíneas seguintes.

a. Analise o código HTML e especifique, no ficheiro **index.js,** as seguintes **variáveis** contantes, logo após a declaração `'use strict'`, de forma a cada uma permita aceder aos elementos DOM necessários:

- → `panelControl` que permite aceder ao elemento `#panel-control`
- → `panelGame` que permite aceder ao elemento `#panel-game`
- → `btLevel` que permite aceder ao elemento `#btLevel`
- → `btPlay` que permite aceder ao elemento `#btPlay`

```
<main>
  <!-- Painel para escolha do nível -->
  <section id="panel-control">
    <h3 class="sr-only">Escolha do Nível</h3>
    <form class="form">
      <fieldset class="form-group">
        <label for="btLevel">Nível:</label>
        <select id="btLevel">···
        </select>
      </fieldset>
      <button type="button" id="btPlay">Iniciar Jogo</button>
    </form>
    <div class="form-metadata">···
    </div>
  </section>
  <!-- Painel onde aparecem as pecas de jogo  -->
  <section>
    <h3 class="sr-only">Peças do Jogo</h3>
    <div id="panel-game">···
    </div>
  </section>
</main>
```



b. Implemente a função **`reset()`**, no ficheiro **index.js,** completando os seguintes passos:

1. Esconda o **`panelGame`**, especificando a propriedade ***display*** com o valor **none;**
2. Implemente o código necessário de forma que o elemento **btPlay** fique desativado. A propriedade que permite esse comportamento é a ***disabled*** com o valor **true**;
3. Invoque a função reset, especificando a instrução de código **reset();** no *scope global* e confirme o resultado no *browser*, certificando-se que não existe nenhum erro de código JavaScript na consola.

c. Confirme, no *browser*, se o resultado tem o aspeto da figura 3.

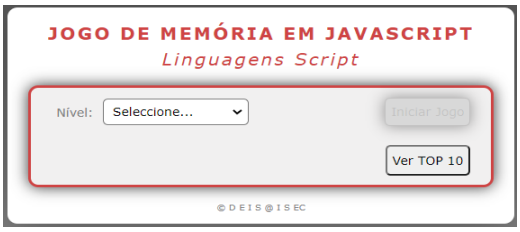


Figura 3 – Início do Jogo

Parte II – Implementação dos DOM Event Listeners

2> Nesta secção, pretende-se efetuar alterações ao aspeto do jogo, quando se clica em alguns elementos existentes no painel de jogo.

Sempre que se selecionar um nível (**btLevel**), o botão de iniciar jogo (**btPlay**) fica ativado, caso contrário, fica desativado, como mostram as figuras seguintes.

Figura 4 - Opção '0' selecionada

Figura 5 - Opção '1' selecionada

a. Para implementar este comportamento, altere a declaração implementada na função **Reset** no qual o botão "Iniciar Jogo" (**#btPlay)** fica **desativado** quando a opção selecionada for "*Seleccione…*", opção cujo valor é **'0'** (podem confirmar no html), caso contrário, se for selecionado um dos níveis de jogo, o elemento #btPlay fica **ativado.**

Para obter o valor da opção selecionada no elemento *select*, deve recorrer à propriedade **`value`**.

b. Também o **panelGame** deverá ficar **visível** quando se seleciona um nível, aplicando para isso o estilo **grid** à propriedade **display** e **escondido** caso contrário.

c. Por fim, implemente um *Event Listener* de forma a que, sempre que houver uma alteração à opção selecionada (evento **change**) no elemento **btLevel**, seja executada a função **reset**.

d. Confirme, no *browser*, se o resultado tem o comportamento pretendido.

3> Tendo em consideração o código das alíneas anteriores, implemente o código necessário de forma que, quando se clica no botão "Iniciar Jogo", o seu texto mude para "Terminar Jogo" e apresente os vários elementos com os dados do jogo, como se apresenta na figura seguinte.

Para obter o comportamento e aspeto da figura 6, complete os passos seguintes.

Figura 6 - Jogo Iniciado

a. Implemente a função `startGame`, de forma a que:

1. Inative o elemento que permite a seleção do nível (*btLevel*)
2. Especifique o texto "Terminar Jogo" ao botão **btPlay**
3. Adicione a class **gameStarted** a todos elementos especificados com a classe `.list-item`. Para isso:
   - → Crie uma variável que obtenha **todos os elementos** especificados com a classe `.list-item` existentes no *painelControl*, com recurso ao `querySelectorAll`.
   - → Com recurso ao **for… of** ou **forEach**, percorra todos os elementos obtidos na alínea anterior de forma a adicionar a **class gameStarted**. Deve usar a propriedade **classList** e o método **add.**

```
<div class="form-metadata">
  <p id="message" role="alert" class="hide" >Clique em Iniciar o Jogo!</p>
  <dl class="list-item left">
    <dt>Tempo de Jogo:</dt>
    <dd id="gameTime">0s</dd>
  </dl>
  <dl class="list-item right">
    <dt>Pontuação TOP:</dt>
    <dd id="pointsTop">0</dd>
  </dl>
  <dl class="list-item left">
    <dt>Pontuação:</dt>
    <dd id="points">0</dd>
  </dl>
  <div id="top10" class="right">
    <button id="btTop">Ver TOP 10</button>
  </div>
</div>
```

b. Implemente a função de nome `stopGame` que:

1. Especifique o texto "Iniciar Jogo" ao botão **btPlay**
2. Ative o elemento que permite seleção do nível (**btLevel**)
3. Por fim, invoque a função **reset**

c. Implemente um *Event Listener* para o elemento **btPlay** de **forma que quando se clica nele:**

- → Se o texto for "Terminar Jogo", invoque a função `stopGame`
- → Caso contrário, invoque a função `startGame`

d. Adicione à função **reset**, o código para que a class **'gameStarted' seja removida** a todos elementos especificados com a classe `.list-item`. Assim, os elementos como o tempo e pontuação de jogo voltam a ficar escondidos, sempre que se cancela ou termina um jogo. Para tal, para todos os elementos `list-item`, invoque o método que remova a class **'gameStarted',** usando a propriedade **classList** e o método **remove**.

e. Confirme, no *browser*, se o resultado tem o comportamento pretendido. Confirme ainda que não existem erros de JavaScript na consola.

4> Para concluir, pretende-se apresentar a mensagem "**Clique em Iniciar Jogo**!" de forma intermitente (aparece/desaparece), sempre que o utilizador clicar no painel de jogo sem o mesmo iniciar. A figura 7 apresenta o resultado desejado.

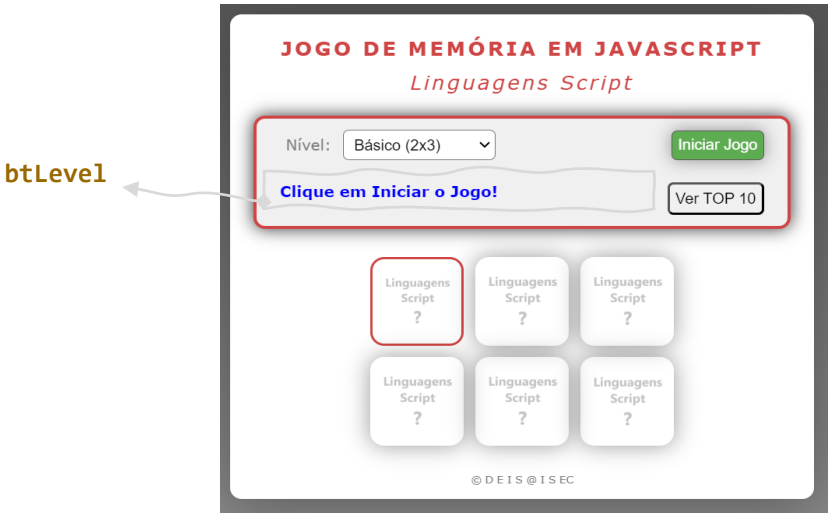


Figura 7 – Mensagem "Clicar em Iniciar jogo"

Para obter este comportamento, complete os passos seguintes.

a. Adicione a variável `message` que permite aceder ao elemento `#message`

b. Na função `startGame`, adicione a classe **hide** ao elemento `message`, para que o elemento com o texto "`Clique em Iniciar Jogo`" não seja visível quando o jogo tiver iniciado. Para isso, recorra à propriedade **classList** e ao método **add** como se apresenta de seguida:

```
message.classList.add('hide');
```

c. Na função `stopGame` remova a classe **hide** que se encontra aplicada ao elemento `message` para que a mensagem volte a poder aparecer.

d. Por fim, especifique o código necessário para que, ao clicar no tabuleiro de jogo, o **panelGame**, a mensagem "**Clique em Iniciar Jogo**!", apareça, e ao voltar a clicar, esta desapareça. Para isso, necessita de adicionar um *event listener* ao elemento **panelGame** e recorra, por exemplo, à propriedade **textContent** ou, ao método *toggle* da **classList**.

Apoio à resolução da ficha:

a. Concentre a declaração das variáveis globais no topo do **index.js**, depois da instrução `'use strict';` por forma a facilitar a sua localização.

b. Variáveis **que não serão alteradas**, devem ser declaradas com **`const`**, caso contrário, declare com **`let`**. Recomenda-se a não utilização da declaração de variáveis com recurso ao **`var`**.

c. Existem várias formas para aceder a um elemento DOM, seja para alterar o seu estado ou para adicionar um *event listener*. Exemplos:

`document.querySelector('#elemento')`	> Permite obter o elemento cujo id é elemento
`document.querySelector('.elemento')`	> Permite obter o 1º elemento com a classe elemento
`document.getElementById('elemento')`	> Permite obter o elemento cujo id é elemento
`document.querySelectorAll('.elemento')`	> Permite obter **todos** os elementos com a classe elemento

a. A **alteração dos estilos** de um elemento pode ser efetuada com recurso à propriedade *`style`*, da seguinte forma:

```
elemento.style.display = 'none';
```

b. **A alteração do texto** de um elemento pode ser efetuada com recurso à propriedade:

```
elemento.textContent = 'JavaScript';
```

c. A **propriedade `classList`** permite aceder às classes associadas a um elemento. Com esta propriedade, e com recurso aos métodos `add()`, `remove()` e `toggle()`, é possível adicionar, remover ou alternar entre duas classes , respetivamente.

```
elemento.classList.add("estilo");
```

d. A sintaxe genérica para definir um *event listener* é a seguinte:

```
elemento.addEventListener(e,function,useCapture)
```

- ***elemento*** – Elemento que se está associar o evento;
- ***e*** – Evento a capturar (ex: click)
- ***function*** – Função a ser executada
- ***useCapture*** – Parâmetro opcional que indica se deve haver encadeamento de eventos

O método *addEventListener* permite anexar um *event handler* a um determinado elemento.

e. Existem várias formas de aplicar o método **addEventListener()**, como apresentado nos exemplos abaixo:

- **Declarando uma função externa** e especificando como argumento o nome dessa função. Esta função pode ser invocada por outros *event listeners* ou outras funções.

  Nota Importante: o nome da função externa é especificado sem parêntesis.

```
const elemento = document.querySelector('h1');
elemento.addEventListener('click', exemplo);

function exemplo() {
     console.log('O elemento h1 foi clicado!');
}

// ou então
function exemplo(event) {
     console.log(`O elemento ${event.target.tagName} foi clicado!`);
}
```

- Recorrendo, **a uma função sem nome** (*anonymous function*) na qual se efetua uma chamada à função externa.

```
const elemento = document.querySelector('h1');
function exemplo(msg) {
     console.log(msg);
}
elemento.addEventListener('click', function () {
     exemplo('Elemento Clicado! Funcao com parametros!');
});

// outra forma…
elemento.addEventListener('click', () =>
       exemplo('Elemento Clicado! Funcao com parametros!'));
```

- Recorrendo, a uma função sem nome (*anonymous function*) na qual implementa o processamento pretendido.

```
const elemento = document.querySelector('h1');

elemento.addEventListener('click', function () {
     console.log('O elemento foi clicado!');
});

// ou então com arrow function
elemento.addEventListener('click', () => {
     console.log('O elemento foi clicado!')
});
```

f. A sintaxe genérica para definir uma estrutura de controlo de repetição com **for..of** é a seguinte:

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

g. A sintaxe genérica para definir uma estrutura de controlo de repetição com **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