> Ficha Prática Nº10 (React – Estados e Interação com utilizador)

Na última ficha prática, foi introduzido o conceito de ***estado*** e como este pode ser aplicado em aplicações React, com recurso recorre aos **Hooks**. Como referido, os Hooks são funções JavaScript reutilizáveis, iniciadas com o prefixo use, tendo sido usado na ultima ficha o Hook **useState,** que permite criação de variáveis de estado, de forma a ser aplicado na aplicação "Jogo de Memória em React". Esta ficha, pretende dar continuidade à transformação do Jogo de Memória de JavaScript para React e nesse contexto, será implementado o painel de jogo de forma dinâmica, bem como, introduzido o conceito de um novo Hook, o **useEffect.**

> Preparação do ambiente

a. Efetue o download e descompacte o ficheiro **ficha10.zip** disponível no *inforestudante*.

NOTA: Os alunos que desejarem, podem continuar a resolução da ficha 9 realizada na última aula.

b. Inicie o *Visual Studio Code* e abra **a pasta no *workspace***.

c. No terminal, digite os seguintes comandos:

- → `npm install` (apenas quem usa ficha10.zip)
- → `npm start` (para iniciar a aplicação)

d. Visualize a página no browser, endereço http://localhost:3000/, que deverá ter o aspeto da figura 3.

Figura 3 – Estado Inicial da Aplicação

Parte II – Exercício1: Criação de Painel Dinâmico

1> Este exercício pretende criar o painel dinâmico, isto é, geração das cartas de acordo com nível de jogo. As figuras seguintes apresentam o que se irá implementar neste exercício. Por questão apenas de confirmar quais as cartas criadas, as cartas têm a classe *flipped* aplicada, embora não seja isso que aconteça realmente no inicio do jogo.

Figura 1 - Geração do Painel dinâmico

Para implementar este processo, será necessário alterar o código em três componentes. Componente **App**, **GamePanel** e **Card**. Assim, implemente os seguintes passos:

a. No componente **App**, foi implementada a função callback **handleLevelchange** que altera a variável de estado ***SelectedLevel*** quando se seleciona o nível, na lista de seleção existente. Será nesta mesma função que se irá preencher o *array* de cartas, de acordo com o nível selecionado (nível básico total de 3 pares, intermédio um total de 6 pares e o avançado com um total de 10 pares). O processo para criação do *array* tem o raciocino semelhando aquando implementação do jogo em javascript, com ligeiras adaptações ao próprio conceito de renderização do react.

Assim, acrescente no topo do componente **App**, a variável de estado **Cards**, em que o estado inicial é um array vazio []. Esta variável irá armazenar todas as cartas que compõem o painel.

Note que para criar os estados dos componentes deve recorrer ao Hook ***useState****.*

b. A geração do painel, tem como base o raciocínio usado na implementação do mesmo jogo em JavaScript, com algumas adaptações ao contexto do React, como por exemplo, o uso de atributos *keys* em cada carta, de forma a que a renderização fique otimizada e impeça nova renderização quando tal não é necessário. Assim, na callback function implemente os seguintes passos:

- → Declare a variável **`numOfCards;`**
- → Recorra a um **`switch`** de forma a que se o valor selecionado no nível for 1 seja atualizado o valor de **numOfCards** para 3, se for '2', o numero de cartas seja 6 e, por fim, se for '3', o numero de catas deve ser 10.
- → De seguida, adicione as seguintes linhas de código, que irão obter um array com os logotipos baralhado e de seguida, obter o número de cartas para que sejam clonadas.

  Note que a função `shuffleArray` é uma função já implementada existente no ficheiro helpers.js e, como tal, deverá efetuar a devida importação. O mesmo acontece com o array

CARDS_LOGOS que se encontra declarado no ficheiro constants.js e terá de ser efetuada a sua importação.

```
const initialCards = shuffleArray(CARDS_LOGOS);
const slicedInitialCards = initialCards.slice(0, numOfCards);
```

→ Visualize o jogo no browser e confirme que não existem erros até ao momento.

→ Copie o seguinte bloco de código e analise o que é efetuado:

```
slicedInitialCards.forEach((card, index) => {
      doubledCardsObjects.push({
        key: `${card}-${index}`,
        id: card,
        name: card
});
doubledCardsObjects.push({
        key: `${card}-${index}-clone`,
        id: `${card}-clone`,
        name: card
      });
});
```

→ Por fim, baralhe novamente o array doubledCardsObjects e atualize o estado do array das cartas.

→ Visualize o jogo no browser e confirme que não existem erros até ao momento.

c. Quando invoca o **GamePanel** no componente App (ver secção do JSX, no *return*), adicione dois atributos:

→ **cards**, que deverá enviar o array das cartas

→ **selectedLevel**, o qual deverá enviar o nível de forma a aplicar a classe correcta quando se gerar o painel.

d. No componente **GamePanel**, receba o valor dos atributos anteriormente definidos via **props** e implemente os seguintes passos:

→ Especifique a variável **gameClasse que deverá ficar com o texto "intermedio" e o nível de jogo seleccionado for 2, se o nível de jogo for 3, deverá ficar co o texto "avançado" caso contrario, a variável deverá ficar com string vazia.**

→ Especifique a className no div cujo id="game", com o valor da variável anteriormente definida.

- Por fim, substitua a invocação das cards existentes pelo código abaixo que irá enviar os dados para a criação das cartas no :

```
{cards.map((card) => (
          <Card key={card.key} card={card} />
))}
```

- Visualize o jogo no browser e confirme que não existem erros até ao momento. Possivelmente constatará que as imagens não aparecem (passo seguinte).

e. Por fim, no componente **Card** necessita de efetuar uma pequena alteração. Repare que anteriormente o atributo *name* era passado directamente, mas neste momento, como pode ver no código acima, é passada o elemento card. Assim, a forma como recebe as props no componente Card, deve ser alterado.

Parte II – Tempo de Jogo

Nesta parte, pretende-se adicionar o tempo de jogo, de forma a que fique com o comportamento da figura abaixo. Para isso, será introduzido o hook *useEffect.*

> *Hooks - useEffect*

O Hook **useEffect** adiciona a capacidade de realizar efeitos secundários a uma função do componente. Basicamente, quando se invoca o Hook, estamos a dizer ao React para executar a função "effect" após surgirem alterações ao DOM, assim, o hook auxilia na forma como se pode ligar com os "side-effects" da

alteração de um determinado estado. O *useEffect* Hook deve ser declarado dentro do componente, de forma a que tenham acesso aos próprios estados e props. Por omissão, o React executa os efeitos, após cada renderização, incluindo a primeira renderização quando a página é apresentada pela primeira vez.

a. De forma a implementar o comportamento pretendido, adicione no componente **App**:

- A variável de estado **timer**, em que o estado inicial deverá ser a constante TIMEOUTGAME, definida no ficheiro contantes. Não se esqueça de efetuar o devido import.

b. De forma a que seja apresentado o tempo no panel de controlo, implemente os seguintes passos:

- Quando invoca o **ControlPanel** no componente App, adicione o atributo **timer,** que deverá passar a variável de estado timer;
- No componente **ControlPanel,** receba também esta propriedade e efetue as alterações para que o valor do timer seja apresentado no browser.
- Visualize o jogo no browser e confirme que é apresentado no tempo de jogo, o TIMEOUTGAME.
- a variável `let timerId = undefined;` abaixo dos imports definidos.

c. Por fim, no componente **App**:

- Adicione o import do `useEffect` hook, do mesmo modo que importou o `useState`.
- Copie o seguinte código que irá implementar o tempo a ser descontado.

```
/**
 * When the component mounts, set an interval for the timer.
 */
useEffect(() => {
  if (gameStarted) {
    timerId = setInterval(() => {
      let nextTimer;
      setTimer((previousState) => {
        nextTimer = previousState - 1;
        return nextTimer;
      });
      if (nextTimer === 0) {
        setGameStarted(false);
      }
    }, 1000);
  } else if (timer !== TIMEOUTGAME) {
    setTimer(TIMEOUTGAME);
  }
  return () => {
    if (timerId) {
      clearInterval(timerId);
    }
  };
}, [gameStarted]);
```