Relatório Linguagens Script – "Minesweeper"

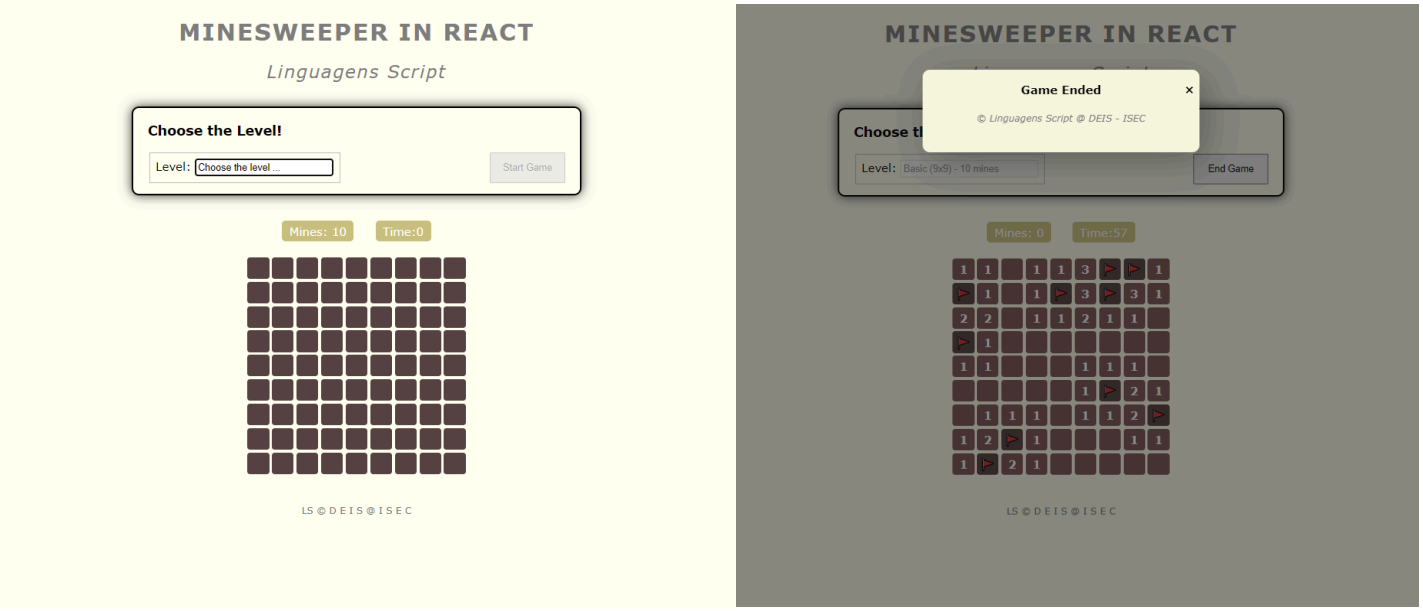

Figura 1 – Interface do jogo minesweeper

Equipa de trabalho

O trabalho prático foi realizado pela seguinte equipa:

- João Pedro Xia
- João Francisco de Matos Claro
- Margarida Francisco Campos

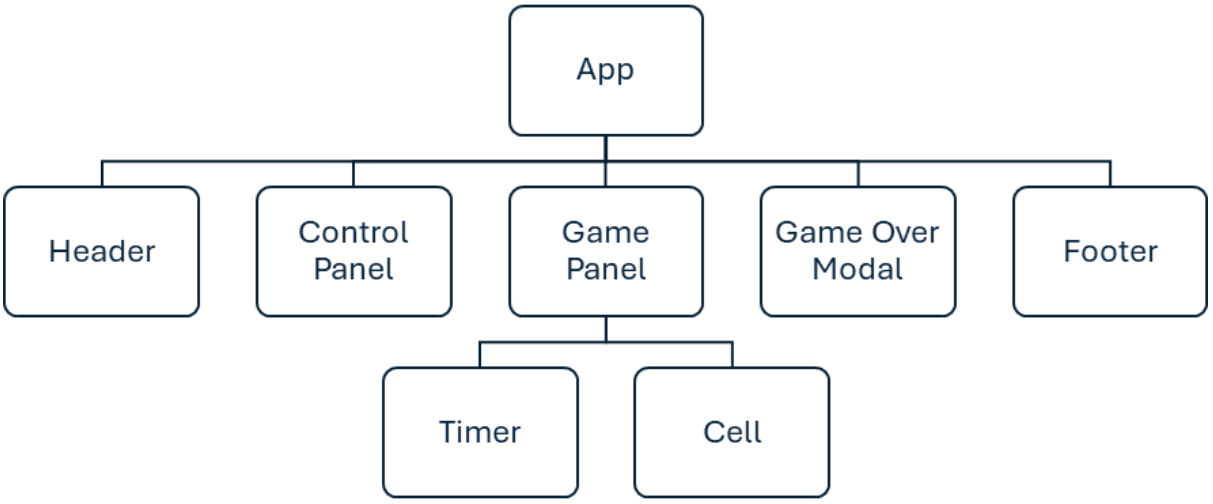

Figura 2- Estrutura do jogo

Resumo

Projeto, em **React JS**, do minesweeper, ou campo minado, permite a utilização de componentes, useState e useEffect, aprendido nas aulas.

Com a maioria das funcionalidades de outros jogos de minesweeper, como mostrar as células vazias próximas do clique do utilizador, mostrar o número de minas nas células adjacentes, e alterar o nível de dificuldade.

Components

App

A componente app é o container principal, gere a interação entre vários componentes filhos, guarda informação a ser partilhada entre componentes.

De seguida temos uma declaração de uma variável que guarda, desde a criação do temporizador até que este é terminado, o identificador do temporizador (*timerId*) utilizando a função embutida do JavaScript, clearInterval(), a qual que recebe o identificador.

Variáveis de estado presentes na app:

- ➔ Em todas existe um set<Nome Variável> que permite ao React receber valores para atualizar a variável de estado em causa.
- ➔ *gameStarted,* um booleano que indica se o jogo começou, inicializado a “false”, ou seja o jogo ainda não começou;
- ➔ *selectedLevel,* uma string que representa o nível selecionado do jogo, inicializada a "0", que representa nível não selecionado.
- ➔ *isGameOverModalOpen,* um booleano que indica se o modal de "game over" está aberto, inicializado a “*false*”, ou seja a componente não está visível para o utilizador, mas está renderizada!

Função ***handleGameStart***

- Alterna o estado de *gameStarted*, se a variável estiver a “*false*”, jogo não começou, utilizamos a função *setGameStarted* fica a “*true”* (inverte *gameStarted* com “!” e agora o jogo começou)

Função ***handleGameOverModalClose***

- Esconde o modal de "*game over*" e redefine *gameStarted* para “*false”*.

Função ***handleGameOver***

- mostra o modal de "*game over*" e limpa o temporizador usando clearInterval

Função ***handleLevelChange***

- Atualiza o nível selecionado baseado no valor do elemento que disparou o evento.

Função ***handleTimer***

- Armazena o identificador do temporizador na variável timerId.

Dados enviados pela App para outros componentes

Comuns a *Control Panel* e *Game Panel*

- *gameStarted={gameStarted}:* Envia o valor da variável de estado *gameStarted* indicando se o jogo está a decorrer.
- *selectedLevel={selectedLevel}:* Envia o valor da variável de estado *selectedLevel* indicando o nível atualmente selecionado.

Control-Panel

- *onGameStart={handleGameStart}:* Envia a função *callback* handleGameStart através do nome onGameStart. É chamada quando o botão de iniciar ou parar o jogo for premido no *ControlPanel*.
- *onSelectedLevel={handleLevelChange}:* Envia a função callback *handleLevelChange* através do nome *onSelectedLevel*. É chamada quando o for alterado o valor do select, simbolizando a mudança do nível selecionado.

Game-Panel

- *onGameOver={handleGameOver}:* Envia a função callback *handleGameOver* através do nome *onGameOver*. É chamada quando o jogo terminar.
- *onTimer={handleTimer}:* Envia a função *handleTimer* como uma prop chamada *onTimer*. É utilizada para que a app receba o identificador do temporizador.

GameOverModal

- *isOpen={isGameOverModalOpen}:* Envia o estado *isGameOverModalOpen* para o *GameOverModal*, indicando se o modal de "game over" deve ser exibido ou escondido.
- *handleClose={handleGameOverModalClose}:* Envia a função *callback handleGameOverModalClose* através do nome *handleClose*. Esta função será chamada para esconder ou mostrar o modal.

ControlPanel

A componente ControlPanel é responsável por fornecer a UI para haver a seleção do nível do jogo e o iniciar ou terminar do jogo.

Na estrutura JSX, um formulário simples com um select, e um botão.

- O ***Select***, um menu com caixa de seleção de opções, tem as opções para os níveis, entre *basic* (básico), *intermediate* (intermédio) e *advanced* (avançado) e a escolha do nível, predefinido. Através da função callback definida na app, envia os valores 1, 2, 3 ou 0 respectivamente.

A opção *disabled={gameStarted}*, alterado se *gameStarted* estiver a "*true*", o jogo já começou, desativando a possibilidade de escolha no campo de seleção, ou se estiver a "*false*", o jogo não está a decorrer, permitindo novamente alterar o nível de jogo.

A propriedade de *onChange={onSelectedLevel}* premiere que, sempre que há uma mudança de valor, a função *callback onSelectedLevel* chama, na app, o *handleLevelChange*, onde é atualizado o nível selecionado baseado no valor selecionado.

- O ***Button*** permite a alteração, pelo utilizador, se pretende começar o jogo ou terminar.

A opção *disabled={selectedLevel === "0"}* Desativa o botão se ainda não tiver sido selecionado nenhum nível, ou seja, se o *selectedLevel* tiver o valor 0.

A propriedade *onClick={onGameStart}* é acionada quando o botão é clicado e chama a função callback *onGameStart*, que na app é o *handleGameStart,* onde é alternado o valor de *gameStarted*.

O texto do botão é alterado se o estado do jogo for alterado. Se já está iniciado ou seja, se *gamestarted = "true",* mostra "*Start Game*", caso contrário, o botão mostra no texto "*End Game*" . Utilizando um operador ternário fica:

{gameStarted ? "End Game" : "Start Game"};

- Em resumo, o *ControlPanel*:

  - ➔ permite selecionar o nível do jogo através de um menu dropdown
  - ➔ Desativa a seleção do nível quando o jogo está a decorrer.
  - ➔ Exibe um botão que permite iniciar ou terminar o jogo
    - ◆ O botão é desativado se nenhum nível for selecionado.
    - ◆ O texto do botão muda dinamicamente entre "*Start Game*" e "*End Game*" com base no estado do jogo.
  - ➔ Chama funções fornecidas (*onGameStart* e *onSelectedLevel*) para alterar o estado do jogo quando são realizadas ações de mudança de nível ou clique no botão.

GamePanel

O GamePanel gere e renderiza o campo do jogo, assim como todas as células associadas e estados do jogo.

Foram criadas duas variáveis globais dentro do *scope* do *GamePanel,* para armazenar o nível do jogo, como texto para alteração dos estilos (*level*), e o número máximo de minas que pode existir a qualquer momento do jogo iniciado (*maxMines*).

Através do *hook useState*, temos três variáveis de estado e as suas funções set<Nome Variável>:

- *mineCount*: armazena a contagem atual de minas no jogo, que inicialmente é definido para 0.
- *grid*: armazena as células do jogo num array unidimensional. Inicialmente é um array vazio.
- *revealedCells*: armazena as células que foram reveladas no decorrer do jogo. Inicialmente é também definido com um array vazio.

Função ***handleMineCount***

- Esta atualiza o estado mineCount com o argumento *isToRemoveMine*. O argumento é um booleano que indica se a contagem de minas deve ser decrementada, caso o valor for verdadeiro, ou incrementado se o valor for falso.

Função ***handleGrid***

- Utilizada para criar o campo de jogo, definindo as minas, as posições das células no campo e o número de minas nas posições adjacentes.
- É iniciado um novo array vazio chamado *newGrid*. Este será preenchido com células para formar o campo do jogo. A variável *maxMines* é atualizada com o número de minas que deve haver na grid, determinado pela função *numMinesOnLevel*. E definem-se as variáveis *width* e *height* com a largura e altura do campo. Estes valores são determinados pela função *boardsize.*
- O ciclo cria as células para o jogo. Cria tantas células quanto o número máximo de células definidas por *width* e *height.* Para cada célula, ele cria uma nova instância da *classe BoardCell* e adiciona ao array *newGrid.* Gerando minas nas primeiras posições do grid, até ao número máximo de minas (*maxMines*).

Função ***shuffleArray***

- Baralha o array newGrid para distribuir as minas de forma aleatória na grid. Depois retorna o array de elementos baralhados e armazena na variável *shuffledArray*.

Função ***calculateMines***

- Percorre cada célula da grid e calcula o número de minas nas células adjacentes. Para fazer isso, ela verifica as oito células ao redor de cada célula e conta quantas delas são minas. Este número é então armazenado na própria célula através da propriedade *numMines*.

Função ***calculatePosition***

- Atribui a cada célula um par de coordenadas (x,y) que representam a sua posição na grid.

As funções *calculateMines* e *calculatePosition* modificam o array *shuffledArray* e retornam o array alterado. Por fim, atualiza a variável de estado Grid com o *shuffledArray.*

Função ***HandleRevealCells***

- Se a célula clicada é uma minada, a função retorna imediatamente e não irá executar as instruções posteriores. Um return cedo para melhorar a performance do programa.

- A variável *cellsToReveal* que armazena um array, inicializada com a célula que foi clicada. É utilizado para percorrer todas as células a serem reveladas.

- Para cada célula neste array, ele invoca a função *checkNeighbors* com o próprio array, a célula atual, a largura e altura da grid para verificar as células adjacentes e adicionar células não minadas que ainda não foram reveladas ao array inicial.

- Por fim, é atualizada a variável de estado *revealedCells*. É acrescentado às células já existentes atualmente as novas células a serem reveladas.

Função ***CheckNeighbors***

- A função recebe o array de células reveladas (*array)*, o índice da célula a procurar as adjacentes (cell), a largura da *grid* e a altura da *grid (width e height)*.

- Se o número de minas na célula atual for diferente de zero, significa que a célula atual tem pelo menos uma mina nas células adjacentes, então a função *CheckNeighbors* termina a validação. Isto garante que não há revelação de células na diagonal.

- *Directions* é um array que representa os oito pontos cardeais possíveis relativos à célula. Cada direção é representada por um par de números [*dx, dy*], onde *dx* é o deslocamento na coordenada *x* e *dy* é o deslocamento na coordenada *y*. Eles podem ser -1, 0 ou 1, dependendo da direção.

- O ciclo percorre cada direção do array directions e ele calcula as novas posições newX e *newY* adicionando os respectivos deslocamentos *dx* e *dy* à posição atual. Guarda também na variável *newCell* o índice da célula vizinha no array através da conta *cell+dx+dy*width*.

- A instrução da validação garante que as células a validar estão dentro dos limites do tabuleiro. As coordenadas *x ou y* têm de ser maior ou igual a 0 para garantir que não estamos a sair da *grid* para a esquerda ou para cima, menor que a largura da *grid* para garantir que não estamos a sair do lado direito da *grid, e* menor que a altura do tabuleiro para garantir que não estamos a sair da *grid* para baixo*.*

- O restante da validação garante que não procuramos uma célula que já foi procurada anteriormente, de modo a não gerar um ciclo infinito de pesquisa. O método *includes* procura por todos os elementos do array se existe uma determinada célula.

- Por último apenas adiciona se a célula a ser procurada não é uma mina.

Função ***handleGameOver***

- É chamada quando o jogo precisa de verificar se terminou, seja porque o jogador clicou numa célula minada ou porque todas as células foram reveladas.

- Caso seja carregado numa mina, a função vai mostrar para o utilizador a posição de todas as minas existentes no campo de jogo.

- Se o utilizador não tiver carregado em nenhuma mina e já tiver revelado todas as posições possíveis, então o jogo também termina.

Hooks ***useEffect***

- Sempre que o valor de *gameStarted* ou o *selectedLevel* são alterados, o array *revealedCells* é esvaziado para um array vazio. Isto é feito para garantir que todas as células da grid estejam escondidas no início de cada jogo.

- É atualizado a variável de estado *mineCount* para o número de minas que deve haver na grid, que é determinado pela função *numMinesOnLevel*, e a variável level com base no *selectedLevel.*

- Isso significa que sempre que o jogador escolher um novo nível de dificuldade ou iniciar um novo jogo, a grid será atualizada consoante a dificuldade, a contagem de minas será redefinida e a variável nível será atualizada.

Renderização

- O componente é dividido em 2 partes principais: *info* e *gamePanel*.

- O *gamePanel*, o container principal, monta todas as células presentes no grid atual.

- A *div "info"* que contém o número de minas restantes (*mineCount*) e o temporizador (Timer). Este é montado se o jogo tiver sido iniciado, e desmontado caso contrário.

- O board tem uma propriedade de onContextMenu que garante que o utilizador não tem o menu por defeito do *browser* quando carrega com o botão direito do rato.

class *BoardCell*

- Classe, em JS uma classe é uma função que é usada como um modelo para a criação de objetos. Neste caso, para criar células no tabuleiro.

- O construtor é um método especial que é automaticamente chamado quando uma nova célula é criada. Neste caso, o construtor recebe apenas um argumento *isMined*.

- Propriedades:

    - *isMined* é definida com o valor do argumento.

    - *isFlagged* é definida como "*false"*. Isto indica que a célula não tem nenhuma marcação

    - *isRevealed* é definida também como "*false*". Isto indica que a célula não está revelada.

    - x / y são definidos a 0. O par representa as coordenadas (x,y) da célula no tabuleiro.

GameOverModal

O GameOverModal é responsável por exibir uma janela quando o jogo termina.

A função GameOverModal recebe três valores por props:

- *isOpen*: um booleano que determina se o modal está mostrado ou não.
- *handleClose*: uma função a ser chamada para esconder ou mostrar a janela.
- *points*: a pontuação que o jogador atingiu no jogo. (não utilizado no contexto do trabalho)

useRef é um hook que é usado para criar uma referência a um elemento DOM ou uma instância de um componente. Neste caso, cria uma referência (ref) para o elemento de diálogo. Isto permite-nos chamar métodos no elemento de diálogo diretamente.

useEffect: o uso do hook para mostrar ou esconder a janela sempre que *isOpen* muda.

Footer

- O footer simplesmente mostra informação de copyrights no rodapé.

Header

- O header mostra alguma informação pertinente ao utilizador. O título da página “Minesweeper in React” e o subtítulo “Linguagens Script”, indicando a razão de ser do projeto.

Timer

- O componente timer cria apenas, com ajuda da função embutida, de setInterval() um intervalo crescente, que marca o tempo de jogo a decorrer.
- Retorna o tempo em segundos e é atualizado a cada 1000ms, o que corresponde a 1 segundo.

Cell

Props da componente cell:

- *id*: o identificador único para cada célula.
- *isMined*: um booleano a indicar se a célula tem mina ou não.
- *numMines*: o número total de minas nas células adjacentes.
- *onReveal*: uma função responsável por revelar as células.
- *revealedCells*: o número de células a serem reveladas.
- *onGameOver*: uma função responsável por lidar com o fim do jogo.
- *onMineCount*: uma função responsável por alterar o número de minas.
- *gameStarted*: um booleano a indicar se o jogo já começou ou não.

Variáveis de estado:

- *isFlag*: uma string a indicar se a célula está marcada ou não. Inicializada com uma string vazia.
- *isRevealed*: um booleano a indicar se a célula está revelada ou não. Inicializada com valor "*False*".

Hooks useEffect

- O hook useEffect coloca todas as células sem marcação quando é começado ou terminado um jogo.
- O segundo hook useEffect está configurado para ser executado sempre que o array revealedCells ou o id mudarem. Revelando as células que não estão mostradas e que estão na variável de estado. Por fim chama a função onGameOver. esta função é responsável por confirmar se o jogo chegou ao fim.

Função ***handleSetFlag***

- É chamada quando o utilizador carrega o botão direito do rato numa célula. Esta verifica a flag de momento na célula e alterna entre vazio (""), com bandeira ("flagged") e possibilidade de bandeira ("possible").

Função ***handleOnClick***

- É chamada quando o utilizador carrega em qualquer célula. No início verifica se foi o botão esquerdo ou direito. No direito, chama a função *handleSetFlag.* Caso tenha sido o esquerdo, verifico se é uma mina e chama a função que faz a revelação da célula e das adjacentes.

Limitações conhecidas

De momento o utilizador pode perder durante o primeiro movimento após iniciar o jogo.

O utilizador não consegue carregar no número da célula revelada para revelar, caso seja possível, as células garantidas com não terem minas.

O Hook useEffect na cell pede inclusão do onGameOver nas dependências, à qual, quando incluído gera a chamada da função a todas as renderizações.

Desafios

O primeiro grande desafio foi o de confirmar as células adjacentes pelo número de minas. Na qual teve que se fazer alteração à maneira como a componente Cell foi criada, pois esta não tem acesso às células adjacentes. Para tal tivemos que “puxar” a verificação para o GamePanel.

O segundo desafio foi para revelar grupos de células vazias, pois este gerava um ciclo infinito. A razão era que as variáveis de estado não são atualizadas durante a execução do ciclo, e como tal, para a verificação, ainda não tinham sido reveladas ou procuradas.

Conclusão

Este trabalho prático tem como objetivo o desenvolvimento de uma aplicação em **React JS**, na qual tivemos a chance de aplicar todos os conhecimentos adquiridos ao longo das aulas. Permitiu-nos desenvolver as capacidades técnicas para dar continuidade a outros projetos em React.