Relatório Linguagens Script – "Minesweeper"

Figura 1 – Interface do jogo (screenshot do jogo)

➢ *Resumo*

Este trabalho prático tem como objetivo o desenvolvimento de uma aplicação utilizando a biblioteca React JS, nos aproveitando da atualização e renderização dos elementos da interface, características cruciais para o desenvolvimento de um jogo como o Minesweeper, que requer atualizações frequentes e precisas do estado do tabuleiro com base nas ações do usuário.

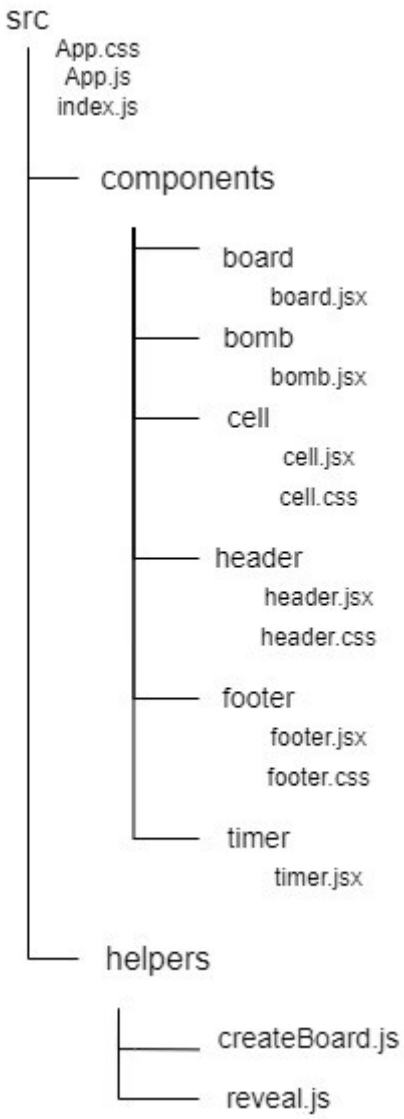
➢ *Equipa de trabalho*

O trabalho prático foi realizado por:

Guilherme Farias Costa – 2022144234;

Thiago Santi Tavares – 2022134334;

➢ *Components*



O jogo a implementar deverá ser acessível através do browser e deverá disponibilizar, pelo menos, as seguintes funcionalidades:

→ Header.jsx

O componente Header contém o menu de seleção de nível e o botão de iniciar jogo. Ele também inclui o componente Timer para exibir o tempo de jogo e o exibe o número de bombas restantes.

Props Recebidas:

selectedLevel: Nível de dificuldade selecionado.

onLevelChange: Função para mudar o nível de dificuldade.

startGame: Função para iniciar o jogo.

isGameRunning: Indica se o jogo está em andamento.

remainingBombs: Indica a quantidade de bombas no tabuleiro.

Funções Principais:

Permite o utilizador escolher o nível de dificuldade, e desabilita a seleção de nível e o botão de iniciar jogo quando o jogo está em andamento, além de mostrar o tempo de jogo, criado pela componente “<Timer />”, e o número de bombas presentes no tabuleiro.

- **Board.jsx**

O componente Board é responsável por gerenciar o tabuleiro do jogo. Ele gera o tabuleiro inicial, atualiza o estado do jogo e gerencia a revelação das células e colocação de bandeiras e interrogações.

Estados Gerenciados:

grid: Armazena o tabuleiro com suas células.

nonMineCount: Contador de células que não são minas e que ainda não foram reveladas.

mineLocations: Localizações das minas.

gameOver: Indica se o jogo terminou.

Funções Principais:

freshBoard: Gera um novo tabuleiro.

revealCell: Revela uma célula, termina o jogo se é encontrado uma mina e atualiza os estados “grid” e “nonMineCount”.

updateFlag: Coloca uma bandeira ou interrogação em uma célula, atualiza os estados “grid” e “setRemainingBombs” e previne a ação padrão do clique direito.

Renderização:

Renderiza as células do tabuleiro.

- **Cell.jsx**

O componente Cell é responsável por apresentar individualmente cada célula no tabuleiro. Trabalhando com a lógica de revelação de células e colocação de bandeiras ou interrogações.

Props Recebidas:

revealCell: Função para revelar a célula.

updateFlag: Função para colocar uma bandeira ou interrogação na célula.

details: Informações da célula como, se está revelada, se tem bandeira, valor numérico presente.

gameOver: Indica se o jogo chegou ao fim.

boardClass: Classe CSS para o tamanho do tabuleiro escolhido pelo utilizador.

Funções Principais:

Cria a célula com o estilo específico para cada situação, bloqueia a revelação de células com bandeiras ou interrogações e alterna o estado da célula (normal, bandeira, interrogação) e revela a célula quando é clicado com o botão esquerdo, com a função handleClick.

→ Timer.jsx

O componente Timer exibe o tempo de jogo, em segundos, desde o início até o fim. Ele some com o fim do jogo.

Props Recebidas:

gameOver: Indica se o jogo terminou.

→ Bomb.jsx

O componente Bomb apenas renderiza o ícoe de uma bomba no tabuleiro. Quando uma célula contendo uma mina é revelada, ou seja, quando a célula é revelada e o seu details.value === “X”, a componente Bomb é renderizada.

Footer.jsx

O componente Footer exibe uma mensagem "Fim de jogo!" quando o jogo termina, basicamente se gameOver for verdadeiro, renderiza a mensagem, mas se for null, não faz nada.

Props Recebidas:

gameOver: Indica se o jogo terminou.

➢ *Limitações conhecidas*

Não houve limitações naquilo que propus a fazer. No entanto, faltaram detalhes que poderiam enriquecer o jogo, como animações, uma função que guardava o melhor tempo do jogador em cada tabuleiro e níveis customizáveis, onde o utilizador poderia escolher o número de linhas, colunas e minas do tabuleiro.

➢ *Desafios*

O principal desafio encontrado foi na criação da opção mais valorizada para este trabalho, que consiste em esvaziar as células vizinhas vazias até encontrar células com indicação de presença de mina em sua adjacência.

O uso das variáveis de estado foram desafiantes e garantiram as funcionalidades mais importantes da aplicação, retendo os dados necessários, como o estado do tabuleiro, o número de células que ainda não foram marcadas, etc. Juntamente com outro hook, o “useEffect”, garantiu a atualização do tabuleiro em tempo real e na inicialização do temporizador.

➢ *Conclusão*

Em conclusão, o trabalho prático foi desafiante e a implementação das funcionalidades e a lógica de revelação das células adjacentes exigiram uma boa compreensão dos conceitos de estado e efeito no React.