

## **Relatório Linguagens Script – “MineSweeper”**



*Figura 1 – Interface do jogo MineSweeper*

### > Resumo

Este Trabalho técnico acompanha o trabalho prático desenvolvido, que tem como objetivo o desenvolvimento de uma aplicação web em React JS do jogo popular ‘MineSweeper’, acreditamos que tenha sido demonstrado o domínio e conhecimento da tecnologia React assim como JS, HTML e CSS. Neste relatório, iremos apresentar o diagrama de componentes da aplicação, descrever as suas funcionalidades e destacar as soluções adotadas durante o desenvolvimento do mesmo.

### > Equipa de trabalho

O trabalho prático foi realizado pela seguinte equipa:

- Rodrigo Gonçalves
- Ricardo Duarte

## > Diagrama de Componentes

Par comecar, apresentamos o diagrama de componentes de forma a facilitar a interpretação da gestão de componentes React necessários à implementação do jogo. Dividimos o nosso projeto em alguns componentes para nos ser mais fácil a manutenção de cada módulo.

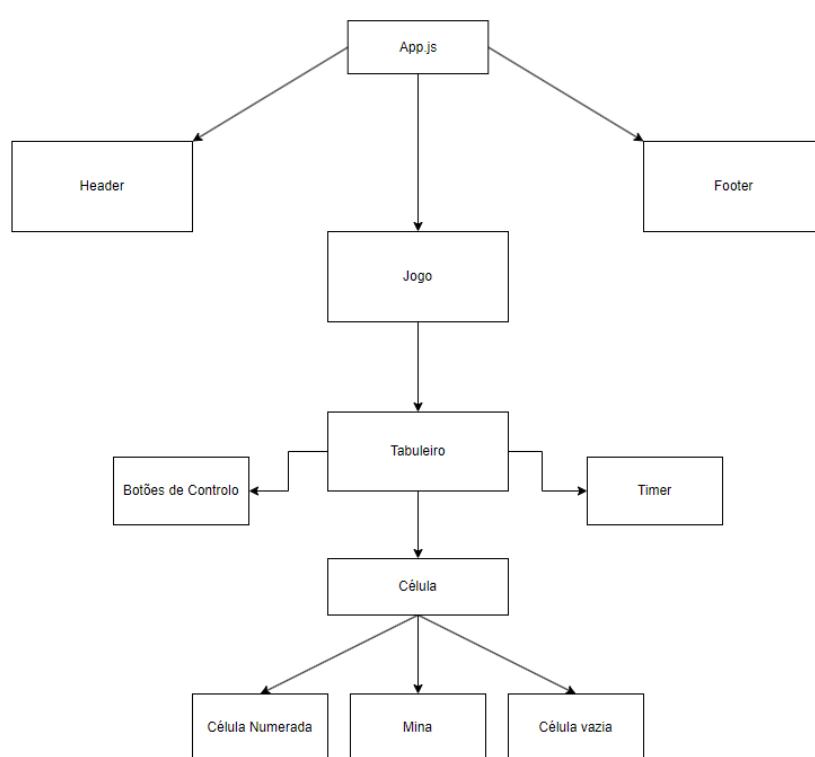


Figura 2 – Diagrama de Componentes

## > Funcionalidades Implementadas

Ao longo do desenvolvimento do jogo fomos tendo algumas dificuldades em implementar algumas funcionalidades, então decidimos focar nas que nos eram mais acessíveis, benéficas e promissoras. Seguem-se então todas as funcionalidades implementadas:

- Expansão de células vazias vizinhas;
- Algarismo indicador do número de minas nas células vizinhas;
- Um clique para marcar a célula com uma flag, dois cliques para marcar como possível mina;
- Temporizador que conta o tempo de jogo;
- Contador de minas restantes no topo do jogo;
- Seletor de nível de jogo;
- Indicador de fim de jogo, seja por vitória ou derrota;

## > Componentes

O componente Jogo em React representa um jogo com diferentes níveis de dificuldade. Ele permite selecionar um nível de jogo, reiniciar o jogo e exibir informações como o número de minas restantes e o tempo decorrido.

```

src > components > Jogo > Jogo.component.jsx > Jogo
1 import React, { useState, useEffect, useCallback } from "react";
2 import Tabuleiro from "../Tabuleiro/tabuleiro.component";
3 import "./style.css";
4
5 function Jogo({ onLevelChange }) {
6   const [settings, setSettings] = useState({
7     height: 9,
8     width: 9,
9     mines: 10,
10   });
11
12   const [mines_conta, setMines] = useState(settings.mines);
13   const [cronometro, setCronometro] = useState(0);
14   const [resultado, setResultado] = useState(null);
15
16   // Função para recomeçar o jogo
17   const restartGame = () => {
18     // Recarrega a página
19     window.location.reload();
20   };
21
22   // Função para lidar com a mudança de seleção
23   const handleSelectChange = useCallback((event) => {
24     const value = event.target.value;
25     setSettings(prevSettings => {
26       let newSettings;
27       switch (value) {
28         case 'basic':
29           newSettings = { height: 9, width: 9, mines: 10, level: 'basic' };
30           break;
31         case 'intermediate':
32           newSettings = { height: 16, width: 16, mines: 40, level: 'intermediate' };
33           break;
34         case 'advanced':
35           newSettings = { height: 30, width: 16, mines: 99, level: 'advanced' };
36           break;
37         default:
38           newSettings = prevSettings;
39           break;
40       }
41       onLevelChange(newSettings.level);
42       // Salva o nível selecionado no localStorage
43       localStorage.setItem("selectedLevel", value);
44       return newSettings;
45     });
46   }, [onLevelChange]);
47

```

Figura 3 - Excerto de código do componente jogo

O componente Tabuleiro Inicia o jogo se este ainda não começou, com um clique revela a célula, se esta for uma mina, termina o jogo como derrota. Se a célula estiver vazia, revela os vizinhos vazios, atualiza o estado do tabuleiro e verifica a vitória.

A função handleRightClick Lida com cliques do botão direito do rato em uma célula:

- Marca a célula com um estado de bandeira("🚩");
- Interrogação ("?") ou vazio.

Atualiza o estado do tabuleiro e verifica a vitória.

Função createNewBoard Cria um novo tabuleiro com minas distribuídas aleatoriamente, Usa getRandomMines para determinar a posição das minas e getNeighbours para atualizar as células adjacentes.

```

src > components > Tabuleiro > tabuleiro.component.jsx > Tabuleiro > constructor > createNewBoard > useCallback() callback > addGridCell > neighbours
  1 import React, { useState, useEffect, useCallback } from "react";
  2 import PropTypes from "prop-types";
  3 import Celula from "../Celulas/celulas.component";
  4 import "./style.css";
  5
  6 const createGridCell = (y, x, isMine) => ({
  7   x,
  8   y,
  9   n: 0,
 10   isMine,
 11   isRevealed: false,
 12   isFlagged: false,
 13   isUnknown: false,
 14   isClicked: false,
 15   get isEmpty() {
 16     return this.n === 0 && !this.isMine;
 17   }
 18 });
 19
 20 function Tabuleiro({ height, width, mines, encontra_minas, tempo_decorrido, fimjogo }) {
 21
 22   const [grid, setGrid] = useState([]);
 23   const [minesCount, setMinesCount] = useState(mines);
 24   const [elapsedTime, setElapsedTime] = useState(0);
 25   const [gameStarted, setGameStarted] = useState(false);
 26   const [gameOver, setGameOver] = useState(false);
 27
 28   const getNeighbours = useCallback((grid, y, x) => {
 29     const neighbours = [];
 30     const currentRow = grid[y];
 31     const prevRow = grid[y - 1];
 32     const nextRow = grid[y + 1];
 33
 34     if (currentRow[x - 1]) neighbours.push(currentRow[x - 1]);
 35     if (currentRow[x + 1]) neighbours.push(currentRow[x + 1]);
 36     if (prevRow) {
 37       if (prevRow[x - 1]) neighbours.push(prevRow[x - 1]);
 38       if (prevRow[x]) neighbours.push(prevRow[x]);
 39       if (prevRow[x + 1]) neighbours.push(prevRow[x + 1]);
 40     }
 41     if (nextRow) {
 42       if (nextRow[x - 1]) neighbours.push(nextRow[x - 1]);
 43       if (nextRow[x]) neighbours.push(nextRow[x]);
 44       if (nextRow[x + 1]) neighbours.push(nextRow[x + 1]);
 45     }
 46
 47     return neighbours;
 48   }, []);

```

Figura 4 – Excerto de Código do componente Tabuleiro

O componente célula tem a função de determinar o valor a ser exibido na célula com base no seu estado:

- Se esta não estiver revelada, retorna `value.flagState` que pode ser “▶” ou “?” ou “null”;
- Quando for revelada e se contiver uma mina, retorna “💣”;
- Se estiver vazia, retorna uma string vazia;
- Se estiver perto de minhas, mostra o número de minas adjacentes na célula.

```
src > components > Celulas > celulas.component.jsx > cellItemShape > isMine
  1 import React from "react";
  2 import PropTypes from "prop-types";
  3 import "./style.css";
  4
  5 function Celulas({ value, onClick, cMenu }) {
  6
  7   const getValue = () => {
  8     if (!value.isRevealed) {
  9       if (value.isFlagged) {
10         return "🚩";
11       } else if (value.isUnknown) {
12         return "?";
13       }
14       return null;
15     } else if (value.isMine) {
16       return "💣";
17     } else if (value.isEmpty) {
18       return "";
19     }
20     return value.n;
21   };
22
23 // Definir a cor baseada no valor da célula
24 const getCorClasse = () => {
25   if (!value.isRevealed) {
26     return "";
27   }
28   switch (value.n) {
29     case 1:
30       return "is-value-1";
31     case 2:
32       return "is-value-2";
33     case 3:
34       return "is-value-3";
35     case 4:
36       return "is-value-4";
37     case 5:
38       return "is-value-5";
39     default:
40       return "is-value-4"; // para numeros que possam aparecer maiores
41   };
42
43 // para aparecer o css suposto, ícones de mina
44 const className =
45   "celula" +
46   (value.isRevealed ? "" : " hidden") +
47   (value.isMine ? " is-mine" : "") +
```

Figura 5 – Excerto de código do componente Célula