Instituto Superior de Engenharia de Coimbra

Licenciatura em Engenharia Informática

Linguagens Script

Trabalho Prático

Ultimate Tic Tac Toe

Bruno Pinto, nº 2021129642

Rúben Ferreira, nº 2021129743

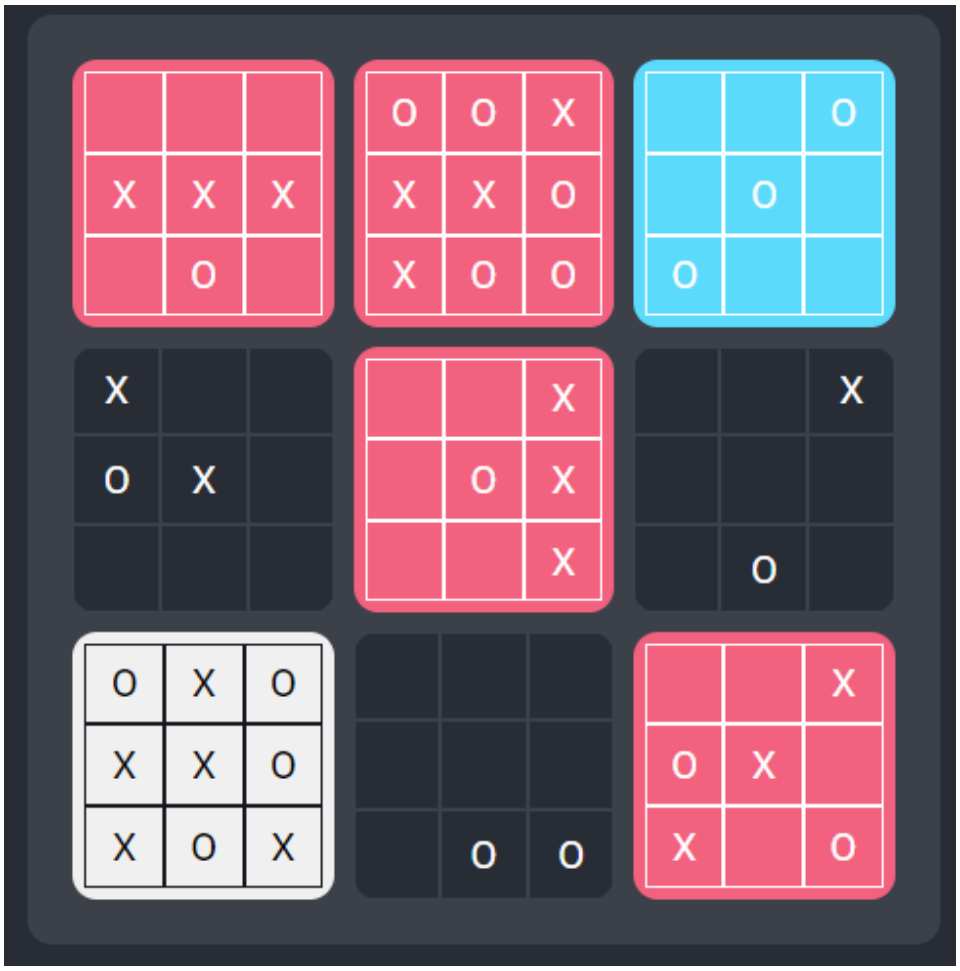
Diogo Ferreira, nº 2021129669

Indice

Introdução

Para o Trabalho Prático da unidade curricular de Linguagens Script é necessário desenvolver o jogo “Ultimate Tic Tac Toe” na liguagem de programação *React*. O jogo é organizado por 9 mini-tabuleiros do jogo do galo, dispostos numa grelha 3x3, sendo que o vencedor do jogo é aquele que vencer 3 jogos dos mini-tabuleiros em linha (seja na horizontal, vertical ou diagonal) do tabuleiro geral.

O IDE usado no trabalho foi o “Visual Studio Code”.



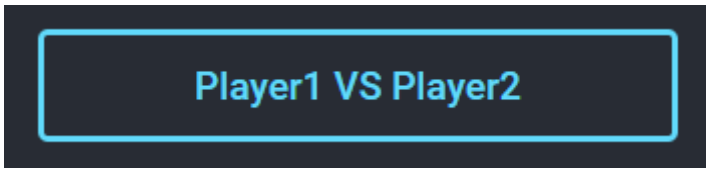
Componentes

Menu principal:

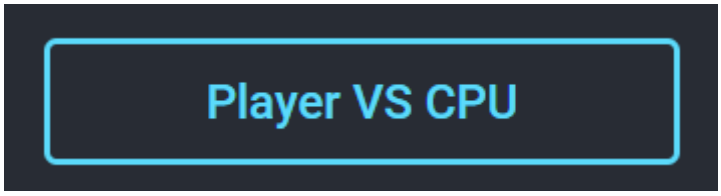
Ao iniciar o Jogo, o usuário depara-se com um menu. Este menu contém:

- 2 Espaços para preencher com o nome/nickname de jogadores.

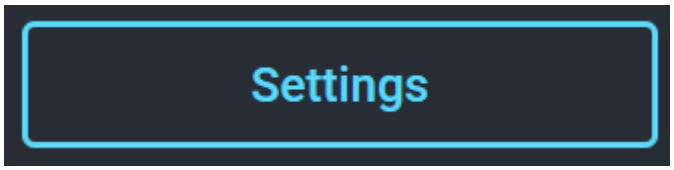
- 1 botão “Player1 VS Player2” que leva o usuário para um tabuleiro de jogo que permite 1 jogador jogar contra outro jogador.



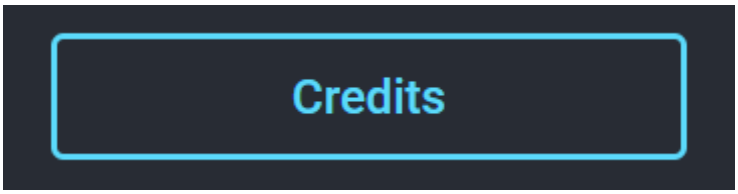
- 1 botão “Player VS CPU” que leva o usuário para um tabuleiro de jogo que permite 1 jogador jogar contra o computador



- 1 botão “Settings” que leva o usuário a um separador onde poderá alterar o tempo máximo de jogo e o tempo máximo para cada jogada



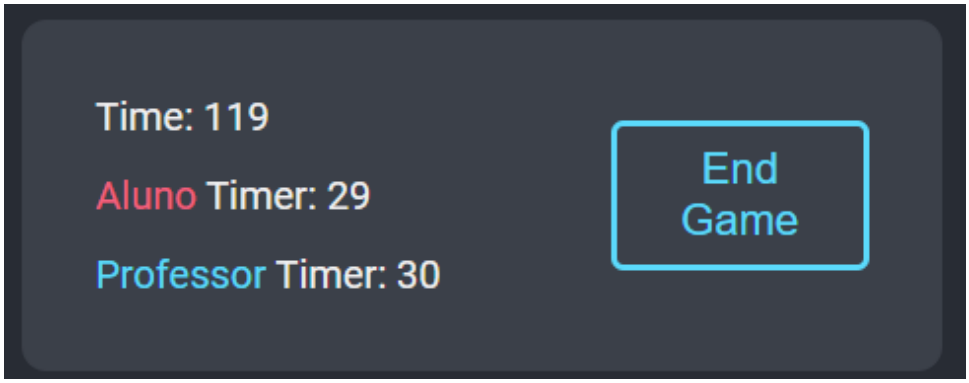
- 1 botão “Credits” que apresenta ao usuário os criadores do jogo.



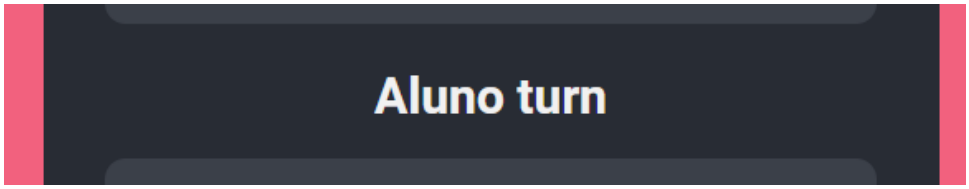
Painel de Jogo

Ao ir para o tabuleiro de jogo (Player1 Vs Player2 ou Player Vs CPU), o usuário depara-se com:

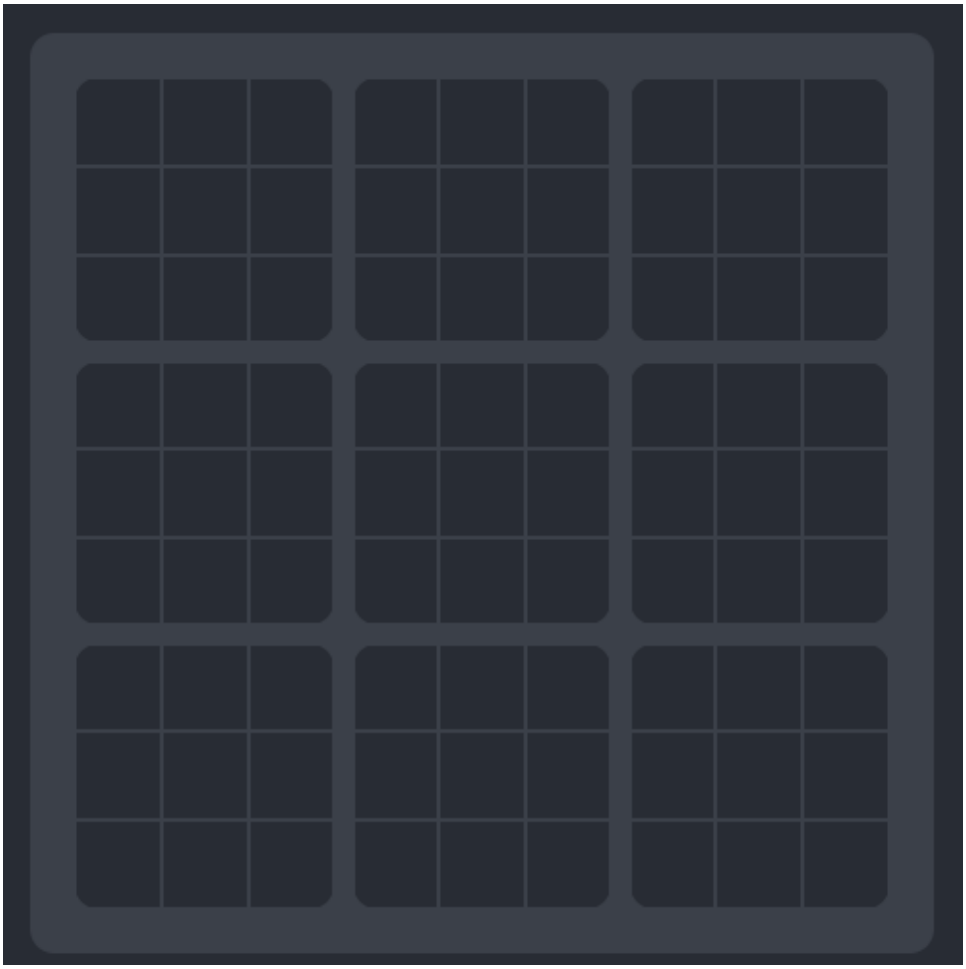
- Um painel que contém:
  - O tempo restante de jogo.
  - O tempo que cada jogador tem para cada jogada.
  - Um botão que termina o jogo e retorna ao menu principal.



- Um texto a indicar de quem será a próxima jogada:



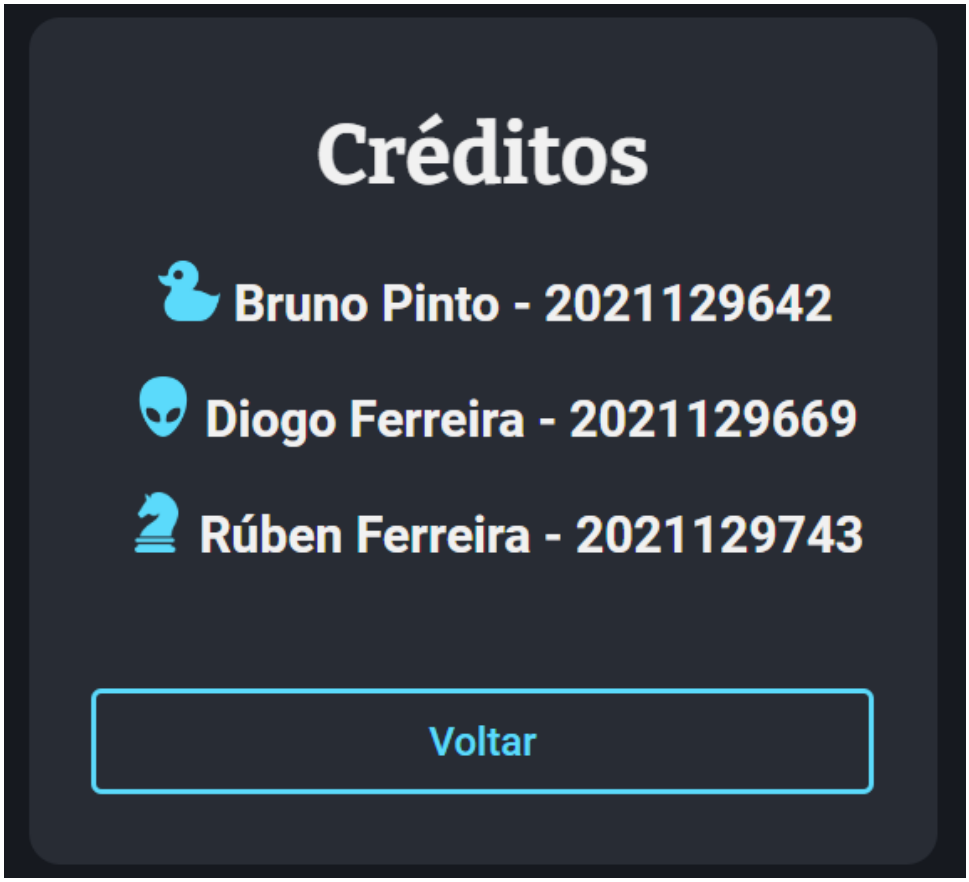
- O tabuleiro onde irá ocorrer o jogo.



Créditos

Ao selecionar o botão “créditos” no menu principal o usuário irá visualizar:

- O nome e número de aluno de todos os criadores do programa
- Um botão “Créditos” que faz voltar ao menu principal.



Funcionalidades

De modo a todo o programa funcionar, abordámos ao longo da execução do trabalho diversas técnicas de programação em React. Iremos agora abordar o código utilizado nas principais partes do trabalho de modo a criar as funcionalidades necessárias.

De modo a recolher os nomes/nicks dos jogadores implementámos: 2 setores de input da seguinte forma:

```
<input type="text" placeholder="player 1" onChange={props.handleName1} />
<input type="text" placeholder="player 2" onChange={props.handleName2} />
```

De maneira a gerar o tabuleiro de jogo, foram implementados 9 “jogos do galo” dispostos numa grelha 3 por 3. De modo a prencher os “mini-tabuleiros” connsiderou-se cada quadricula dos jogos do galo como uma “cell”. Estas estruturas são geradas da seguinte forma:

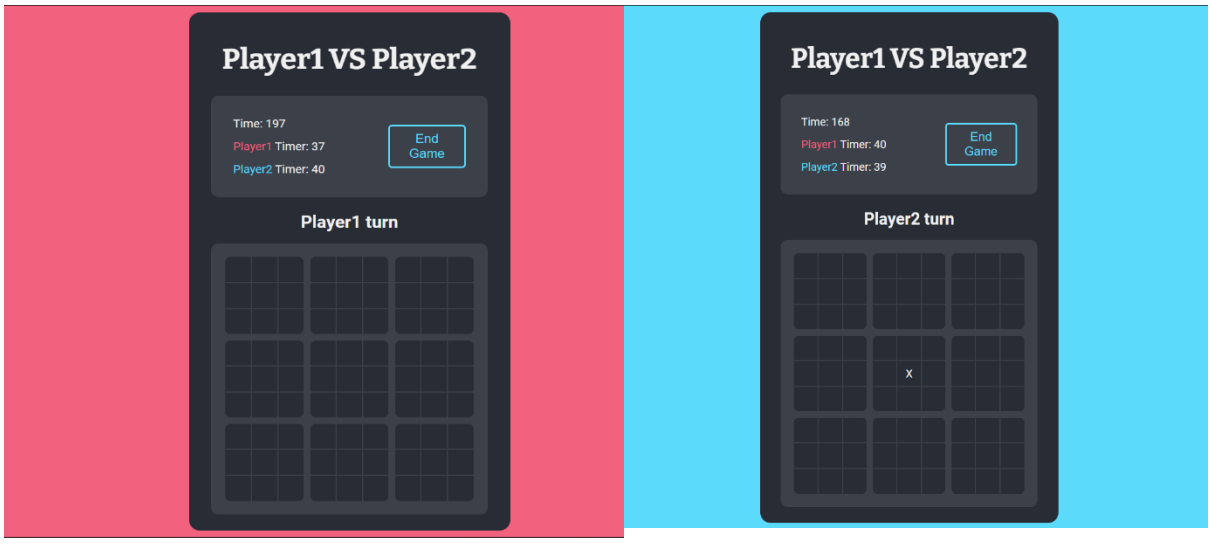
Grelha 3x3:

```
var renderBoards = () => {
  return Array.from({ length: 3 * 3 }, (_, index) => (
    <Board
      key={index + "_board"}
      index={index}
      handleSymbol={handleSymbol}
      getSymbol={getSymbol}
      addBoard={addBoard}
      handleScore={handleScore}
    />
  ));
};
```

Cells de jogo:

```
const renderCells = () => {
  return Array.from({ length: 3*3 }, (_, index) => (
    <Cell
      key={index+'_cell'}
      index={index}
      handleSymbol={props.handleSymbol}
      getSymbol={props.getSymbol}
      handlePlay={handlePlay}
      isWon={won}
    />
  ));
};
```

De modo a proporcionar aos jogadores uma mais rápida e simples perceção de quem é a jogar, fizemos com que o background do jogo troque de cor de acordo com quem é a jogar. Associamos o Player1 à cor vermelha, e o Player2 à cor azul. O resultado é observável nas seguintes imagens:



Esta funcionalidade foi implementada através do seguinte código:

Em ‘App.js’:

```
<div className="container">

<style>{`body { background-color: ${bodyColor}; }`}</style>
```

```
<GamePanel
  type={selectedType}
  handleGameStart={handleGameStart}
  setShowModal={setShowModal}
  setModalContent={setModalContent}
  Pl1={name1}
  Pl2={name2}
  setColor={setBodyColor}
  time={time}
  ptime={ptime}
  gameStarted={gameStarted}
/>
```

Em ‘game-panel.jsx’:

```
useEffect(() => {
  if (getSymbol() === "X") {
    const newColor = '#f15f7f';
    props.setColor(newColor);
  }
  else{
    const newColor = '#61dafb';
    props.setColor(newColor);
  }
});
```

Durante a execução do jogo, o tempo de jogo e os tempos de cada jogada são determinantes no resultado do jogo. Caso o tempo de jogo termine, ganha o jogador com mais mini-tabuleiros ganhos. Caso um jogador deixe terminar o tempo da sua jogada, o adversário ganha instantaneamente.

De modo a implementar o código que controla os temporizadores, utilizámos a seguinte estratégia:

Em game-panel.jsx:

Mudanças de tempo:

```
useEffect(() => {

  const newIntervalId = setInterval(() => {
    setTime((prevTime) => prevTime - 1);

    if (symbol === "X") {
      setPl2Time(pltime);
      setPl1Time((tempo) => (tempo > 0 ? tempo - 1 : 0));
    }
    else{
      setPl1Time(pltime);
      setPl2Time((tempo) => (tempo > 0 ? tempo - 1 : 0));
    }
  }, 1000);

  return () => {
    clearInterval(newIntervalId);
  };
});
```

Vitórias baseadas em tempo:

```
useEffect(() => {

  if(time === 0){

    if(pl1Score > pl2Score){
      props.setShowModal(true);
      props.setModalContent({ title: props.Pl1+'Won', message: 'Score: '+pl1Score, won: 'PX' });

    } else if(pl2Score > pl1Score){
      props.setShowModal(true);
      props.setModalContent({ title: props.Pl2+'Won', message: 'Score: '+pl2Score, won: 'PX' });

    } else {
      props.setShowModal(true);
      props.setModalContent({ title: 'It\'s a Draw', message: props.Pl1+' and '+props.Pl2+' had the same score', won: 'PD' });

    }

  } else if(pl1Time === 0){
    props.setShowModal(true);
    props.setModalContent({ title: props.Pl2+'Won', message: props.Pl1+' ran out of time', won: 'PO' });

  } else if(pl2Time === 0) {
    props.setShowModal(true);
    props.setModalContent({ title: props.Pl1+'Won', message: props.Pl2+' ran out of time', won: 'PX' });

  } else {

    switch (helpers.check(board)) {
      case "X":
        props.setShowModal(true);
        props.setModalContent({ title: props.Pl1, message: 'Score: '+pl1Score, won: 'PX' });
      break;
```

(...)

```
    case "O":
      props.setShowModal(true);
      props.setModalContent({ title: props.Pl2, message: 'Score: '+pl2Score, won: 'PO' });
    break;

    case "D":

    console.log(pl1Score);
    console.log(pl2Score);

    if(pl1Score > pl2Score){
      props.setShowModal(true);
      props.setModalContent({ title: props.Pl1+'Won', message: 'Score: '+pl1Score, won: 'PX' });
    } else if(pl2Score > pl1Score){
      props.setShowModal(true);
      props.setModalContent({ title: props.Pl2+'Won', message: 'Score: '+pl2Score, won: 'PX' });
    }
  break;

  default:
  break;
}
```

De maneira a concluir o jogo, necessitamos de código que verifique o que os jogadores colocam em cada "mini-tabuleiro" e também de código que verifique se algum jogador completou um "3 em linha" com 3 mini-tabuleiros.

Implementámos essas verificações desta forma:

```
const check = (board) => {

  console.log('verificar...');

  // vitoria do X
  if(checkBoard(board, 'X'))
   return 'X';

  // vitoria do O
  if(checkBoard(board, 'O'))
   return 'O';

  // draw
  if(checkDraw(board))
    return 'D';

  return ' ';
};
```

```
const checkBoard = (board, symb) => {

  // horizontais
  if(board[0] === symb && board[0] === board[1] && board[1] === board[2])
    return true;
  if(board[3] === symb && board[3] === board[4] && board[4] === board[5])
    return true;
  if(board[6] === symb && board[6] === board[7] && board[7] === board[8])
    return true;

  // verticais
  if(board[0] === symb && board[0] === board[3] && board[3] === board[6])
    return true;
  if(board[1] === symb && board[1] === board[4] && board[4] === board[7])
    return true;
  if(board[2] === symb && board[2] === board[5] && board[5] === board[8])
    return true;

  // diagonais
  if(board[0] === symb && board[0] === board[4] && board[4] === board[8])
    return true;
  if(board[2] === symb && board[2] === board[4] && board[4] === board[6])
    return true;

  return false;

};

const checkDraw = (board) => {

  for(let i=0; i < 9; i++)
    if(board[i] !== 'X' && board[i] !== 'O')
      return false;
  return true;

};
```

Dificuldades

Durante a execução do trabalho em grupo, deparámo-nos com algumas dificuldades. Muitas dessas dificuldades são características de qualquer trabalho em equipa e/ou projeto de programação em qualquer linguagem.

De entre essas dificuldades iremos realçar as mais impactantes no projeto:

Gestão de tempo

Sendo um trabalho em grupo, todos os elementos concordaram que a grande maioria do projeto fosse feito em simultâneo. Para isso, recorremos a uma extensão do Visual Studio Code chamada “Live Share”, que permite a todos os membros do grupo se juntarem a uma sessão criada por um deles, de modo a

todos editarem os mesmos ficheiros simultaneamente, e reverem o trabalho feito por todos.

No entanto, trabalhar em simultâneo atrai um problema: Agendamentos.

De modo a todos entrarem na sessão de trabalho era necessário a disponibilidade de todos os membros do grupo num mesmo horário, o que nem sempre se conseguia.

Criação do código para gestão do tempo de jogo

Ao longo do desenvolvimento do trabalho, a tarefa que mais tempo nos consumiu foi a criação do código para gestão do tempo de jogo. As primeiras abordagens que tentámos não resultavam devido a alguns problemas de lógica e de invocação de funções e parâmetros. Só após algumas tentativas e pensamento coletivo chegámos à implementação atual, que cumpre os requisitos.

Conclusão

O desenvolvimento de um projeto como este pode ser bastante desafiador para programadores em formação, no entanto é um desafio saudável, que testa os nossos conhecimentos, fortalece o trabalho em equipa e permite uma abordagem mais real do tema, o que facilita a aprendizagem.