Velha Mania — Documentação do Projeto

Versão do documento: 1.0.1

Apresentação

Grupo responsável:

- Luiz Eduardo Jardim de Medeiros
- Enzo Kurt Sales Almeida
- Tiago Oliveira Castro
- Gustavo Henrique Teixeira Viana
- Gustavo Moreira

Visão geral do projeto:

Velha Mania é um aplicativo mobile desenvolvido em React Native que implementa o clássico Jogo da Velha. O objetivo do trabalho é demonstrar conhecimentos em desenvolvimento mobile com React Native: componentes, hooks, navegação por estado, gerenciamento simples de ativos e persistência local (AsyncStorage). A aplicação possui tela splash, menu principal, tela de jogo, placar persistente, tela de "Sobre" e comandos para reiniciar partida e resetar placar.

1. Apresentação

1.1. Descrição do App

O **Velha Mania** é um aplicativo mobile desenvolvido em **React Native (Expo)** que traz uma versão digital e moderna do tradicional jogo da velha. O objetivo é proporcionar uma experiência simples, divertida e acessível, tanto para partidas entre dois jogadores locais quanto para futuras expansões com modos contra IA e histórico de partidas.

1.2. Objetivos

- Proporcionar uma experiência de jogo intuitiva e rápida.
- Explorar os fundamentos do desenvolvimento mobile utilizando React Native.
- Aplicar conceitos de armazenamento local, navegação e interface responsiva.
- Servir como projeto acadêmico demonstrando boas práticas de desenvolvimento front-end.

1.3. Justificativa

O jogo da velha foi escolhido por sua simplicidade e popularidade, permitindo focar mais nas práticas de desenvolvimento e documentação do software. Além disso, é um excelente ponto de partida para aplicar conceitos de arquitetura, persistência de dados e interação usuário-interface.

---

2. O Projeto

2.1. Apresentação da Solução Proposta

A solução consiste em um aplicativo mobile com:

- Interface minimalista e amigável.
- Possibilidade de jogar em dupla no mesmo dispositivo.
- Armazenamento do histórico de partidas usando **AsyncStorage**.
- Pontuação e alternância automática entre os jogadores.
- Possibilidade futura de jogar contra uma IA simples (Minimax) e uso de banco SQLite.

2.2. Arquitetura e Tecnologias Utilizadas

- **Linguagem**: JavaScript (React Native)
- **Framework**: Expo
- **Armazenamento Local**: AsyncStorage
- **Estilo**: Tailwind CSS

- **Gerenciamento de Estado**: Hooks (useState, useEffect)
- **Plataforma**: Android / iOS
- **Arquitetura**: Componentizada e funcional

---

3. Especificações da Solução

3.1. Classificação dos Requisitos

Os requisitos foram divididos em funcionais, não funcionais e de domínio.

3.2. Identificação dos Usuários

- **Jogador Local**: Usuário que utiliza o aplicativo para jogar partidas de velha offline, em um único dispositivo.

3.3. Funcionalidades da Solução

- Iniciar partidas entre dois jogadores.
- Exibir mensagens de vitória ou empate.
- Reiniciar partidas.
- Registrar e exibir histórico de partidas anteriores.
- Controlar pontuação acumulada de cada jogador.

3.4. Requisitos Funcionais

- RF01: O sistema deve permitir que dois jogadores disputem uma partida.
- RF02: O sistema deve identificar quando um jogador vence ou ocorre empate.
- RF03: O sistema deve exibir o placar de vitórias.
- RF04: O sistema deve permitir reiniciar a partida.
- RF05: O sistema deve salvar e carregar histórico de partidas localmente.

3.5. Requisitos Não Funcionais

- RNF01: A interface deve ser responsiva e intuitiva.

- RNF02: O sistema deve funcionar offline.
- RNF03: O tempo de resposta para cada ação não deve exceder 1 segundo.
- RNF04: O aplicativo deve ser compatível com Android e iOS.

3.6. Requisitos de Domínio

- RD01: O jogo segue as regras tradicionais do jogo da velha 3×3.
- RD02: Dois jogadores alternam turnos marcando X e O.

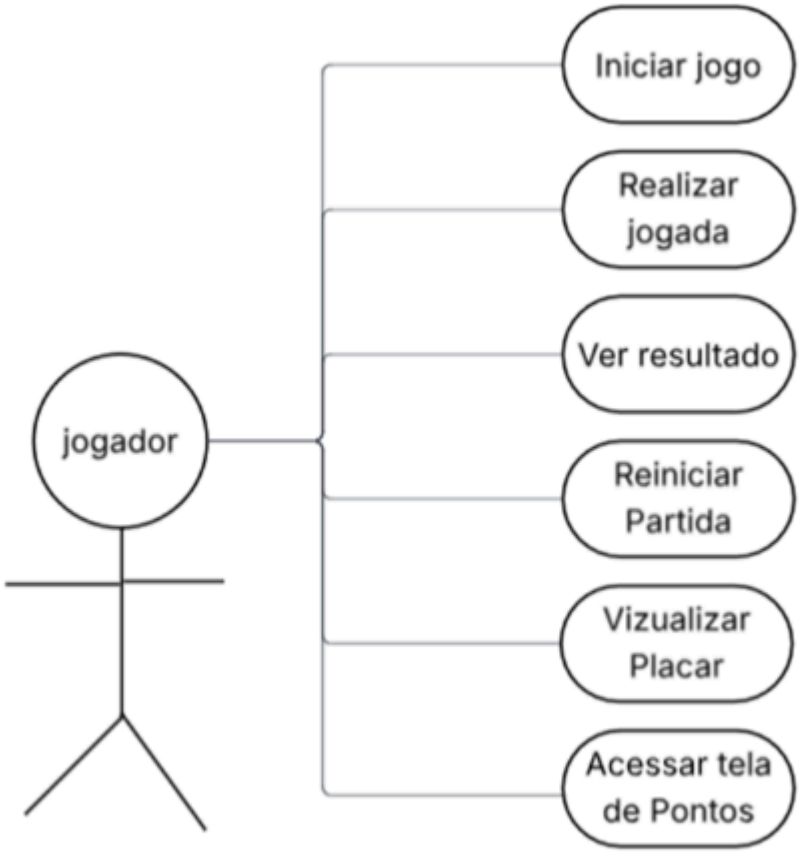
3.7. Diagramas de Caso de Uso

Caso de Uso Principal: Jogar Partida

Atores: Jogador 1

Fluxo Principal:

1. Jogadores iniciam a partida.
2. Alternam jogadas até que haja vitória ou empate.
3. O sistema apresenta resultado e atualiza o histórico.



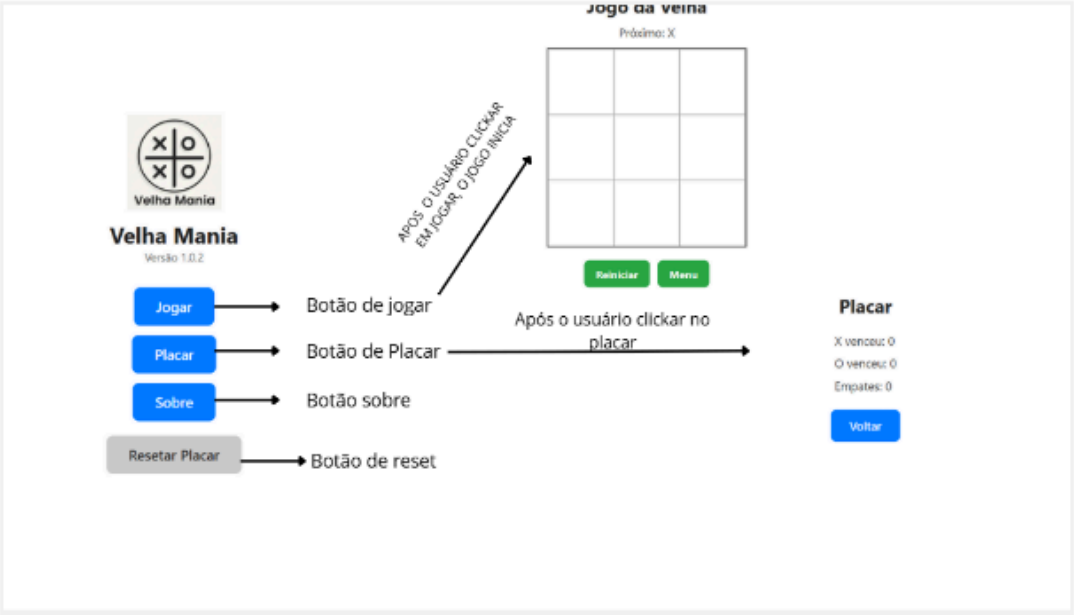
3.8. Projeto da Base de Dados

Para a versão inicial foi utilizado apenas **AsyncStorage** para persistir dados locais.

---

4. O Software Desenvolvido

O aplicativo foi construído com base em uma estrutura componentizada e utiliza estados para controlar o fluxo da partida. O código foi organizado em componentes para tabuleiro, placar e histórico. A navegação e estilização seguem padrões modernos, garantindo boa usabilidade. A persistência com AsyncStorage assegura que o histórico seja mantido mesmo após fechamento do app.



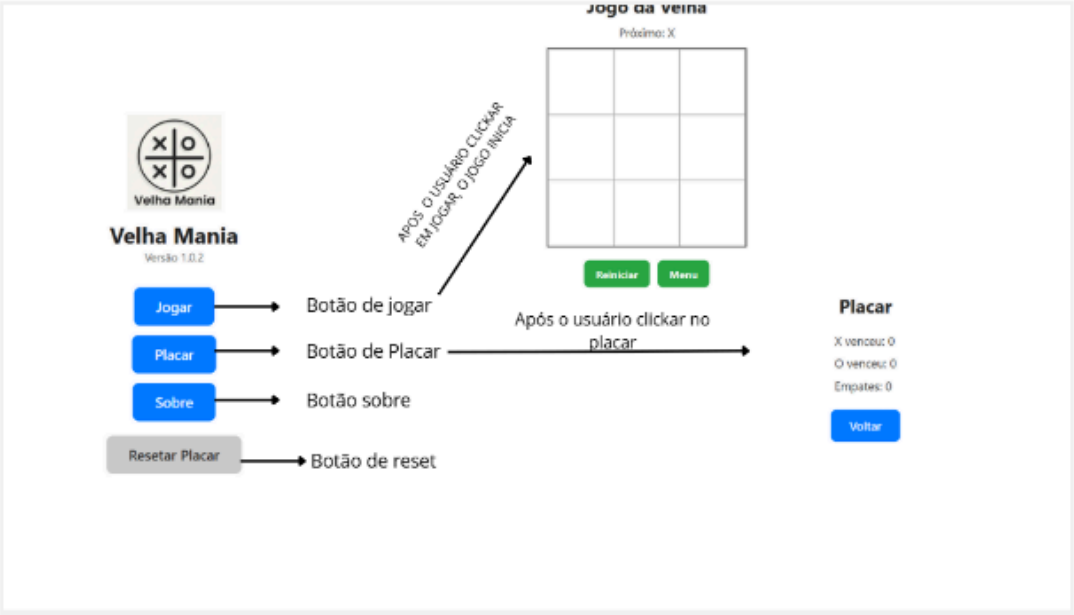
5. Apêndice

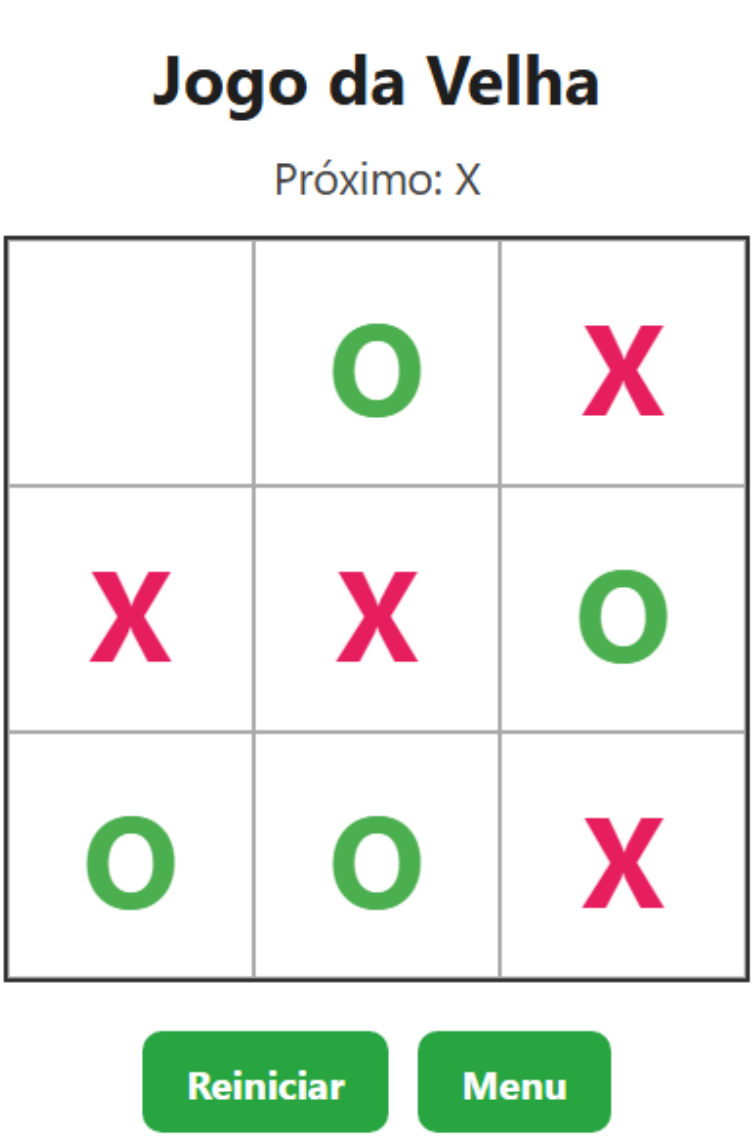
Sugestões para evoluções futuras:

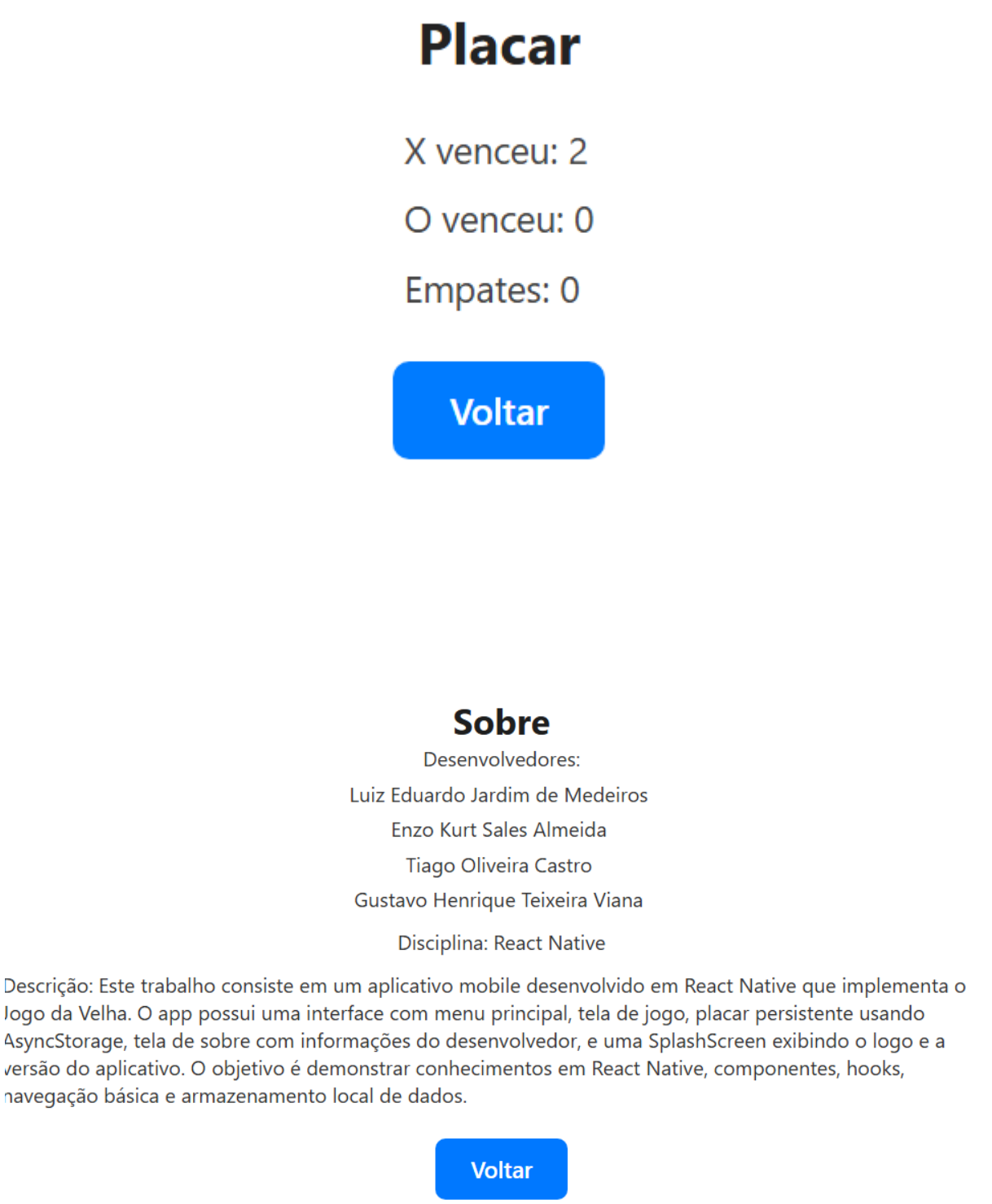
- Implementar IA utilizando o algoritmo **Minimax**.
- Adicionar integração com SQLite.
- Permitir partidas online via WebSocket ou API.
- Melhorar a interface com animações e feedbacks visuais.

6. Anexos

- Prints das telas principais.


Jogo da Velha
Próximo: X
Velha Mania
Velha Mania
Versão 1.0.2
APÓS O USUÁRIO CLICKAR EM JOGAR, O JOGO INICIA
Reiniciar
Menu
Jogar
Placar
Sobre
Resetar Placar
Botão de jogar
Botão de Placar
Botão sobre
Botão de reset
Após o usuário clickar no placar
Placar
X venceu: 0
O venceu: 0
Empates: 0
Voltar


Jogo da Velha
Próximo: X
O
X
X
X
O
O
O
X
Reiniciar
Menu



- Código-fonte do aplicativo.

```
import React, { useState, useEffect } from 'react';
import {
  SafeAreaView,
  StyleSheet,
  Text,
  View,
  TouchableOpacity,
  Alert,
  Image,
  ActivityIndicator,
} from 'react-native';
import AsyncStorage from '@react-native-async-storage/async-storage';

// --- Config ---
const APP_NAME = 'Velha Mania';
const APP_VERSION = '1.0.2';
const STORAGE_KEY = '@velharn:state';

const winningCombos = [
  [0, 1, 2], [3, 4, 5], [6, 7, 8],
  [0, 3, 6], [1, 4, 7], [2, 5, 8],
  [0, 4, 8], [2, 4, 6]
];

// --- Square componente ---
const Square = ({ value, onPress }) => (
  <TouchableOpacity style={styles.square} onPress={onPress} activeOpaci
ty={0.7}>
    <Text style={[styles.squareText, value === 'X' ? styles.playerX : styles.pl
ayerO]}>
      {value}
    </Text>
  </TouchableOpacity>
);

// --- App ---
export default function App() {
  // navegação simples por estado (sem react-navigation)
```

```
  const [screen, setScreen] = useState('splash'); // splash | home | game | s
cores | about
  const [loading, setLoading] = useState(true);

  // estado do jogo
  const [board, setBoard] = useState(Array(9).fill(null));
  const [isXNext, setIsXNext] = useState(true);
  const [winner, setWinner] = useState(null);

  // placar persistente
  const [scores, setScores] = useState({ winsX: 0, winsO: 0, ties: 0 });

  // Carrega armazenamento e mostra splash
  useEffect(() => {
    const init = async () => {
      try {
        const raw = await AsyncStorage.getItem(STORAGE_KEY);
        if (raw) {
          const parsed = JSON.parse(raw);
          if (parsed.scores) setScores(parsed.scores);
        }
      } catch (err) {
        console.warn('Erro lendo storage:', err);
      } finally {
        // mantém splash por 1.8s, depois vai pro menu
        setTimeout(() => {
          setLoading(false);
          setScreen('home');
        }, 1800);
      }
    };
    init();
  }, []);

  // calcula vencedor sempre que o tabuleiro muda
  useEffect(() => {
    const calc = calculateWinner(board);
    if (calc) setWinner(calc);
```

```
    else if (board.every((s) => s !== null)) setWinner('Empate');
    else setWinner(null);
  }, [board]);

  // reage quando houver vencedor: atualiza placar e salva
  useEffect(() => {
    if (!winner) return;

    const applyResult = async () => {
      const newScores = { ...scores };
      if (winner === 'Empate') newScores.ties += 1;
      else if (winner === 'X') newScores.winsX += 1;
      else if (winner === 'O') newScores.winsO += 1;

      setScores(newScores);

      try {
        await AsyncStorage.setItem(
          STORAGE_KEY,
          JSON.stringify({ version: APP_VERSION, scores: newScores, lastPlay
ed: new Date().toISOString() })
        );
      } catch (err) {
        console.warn('Erro salvando placar:', err);
      }

      // mostra alerta com opção de reiniciar imediatamente
      if (winner === 'Empate') {
        Alert.alert('Fim de Jogo', 'Deu Empate!', [{ text: 'Novo Jogo', onPress: r
esetBoard }], { cancelable: false });
      } else {
        Alert.alert('Fim de Jogo', `O jogador ${winner} venceu!`, [{ text: 'Novo
Jogo', onPress: resetBoard }], { cancelable: false });
      }
    };

    applyResult();
    // eslint-disable-next-line react-hooks/exhaustive-deps
```

```
  }, [winner]);

  const calculateWinner = (currentBoard) => {
    for (let i = 0; i < winningCombos.length; i++) {
      const [a, b, c] = winningCombos[i];
      if (currentBoard[a] && currentBoard[a] === currentBoard[b] && current
Board[a] === currentBoard[c]) {
        return currentBoard[a];
      }
    }
    return null;
  };

  const handlePress = (index) => {
    if (board[index] || winner) return; // bloqueia se já preenchido ou fim de j
ogo
    const newBoard = board.slice();
    newBoard[index] = isXNext ? 'X' : 'O';
    setBoard(newBoard);
    setIsXNext((p) => !p);
  };

  const resetBoard = () => {
    setBoard(Array(9).fill(null));
    setIsXNext(true);
    setWinner(null);
  };

  const resetScores = async () => {
    const base = { winsX: 0, winsO: 0, ties: 0 };
    setScores(base);
    try {
      await AsyncStorage.setItem(STORAGE_KEY, JSON.stringify({ version: A
PP_VERSION, scores: base, lastPlayed: null }));
      Alert.alert('Ok', 'Placar reiniciado.');
    } catch (err) {
      console.warn('Erro resetando placar:', err);
    }
```

```
  };

  // status mostrado na UI
  const getStatus = () => {
    if (winner && winner !== 'Empate') return `Vencedor: ${winner}`;
    if (winner === 'Empate') return 'Resultado: Empate';
    return `Próximo: ${isXNext ? 'X' : 'O'}`;
  };

  // ---------- RENDERIZAÇÕES (telas) ----------
  if (loading) {
    return (
      <SafeAreaView style={styles.container}>
        <ActivityIndicator size="large" />
      </SafeAreaView>
    );
  }

  if (screen === 'splash') {
    return (
      <SafeAreaView style={styles.container}>
        <ImagePlaceholder />
        <Text style={styles.title}>{APP_NAME}</Text>
        <Text style={styles.subtitle}>Versão {APP_VERSION}</Text>
      </SafeAreaView>
    );
  }

  if (screen === 'home') {
    return (
      <SafeAreaView style={styles.container}>
        <ImagePlaceholder />
        <Text style={styles.title}>{APP_NAME}</Text>
        <Text style={styles.subtitle}>Versão {APP_VERSION}</Text>

        <TouchableOpacity style={styles.menuButton} onPress={() => { resetB
oard(); setScreen('game'); }}>
          <Text style={styles.menuButtonText}>Jogar</Text>
```

```
      </TouchableOpacity>

      <TouchableOpacity style={styles.menuButton} onPress={() => setScreen('scores')}>
        <Text style={styles.menuButtonText}>Placar</Text>
      </TouchableOpacity>

      <TouchableOpacity style={styles.menuButton} onPress={() => setScreen('about')}>
        <Text style={styles.menuButtonText}>Sobre</Text>
      </TouchableOpacity>

      <TouchableOpacity style={[styles.menuButton, { backgroundColor: '#ccc', marginTop: 18 }]} onPress={resetScores}>
        <Text style={[styles.menuButtonText, { color: '#333' }]}>Resetar Placar</Text>
      </TouchableOpacity>
    </SafeAreaView>
  );
 }

 if (screen === 'scores') {
  return (
    <SafeAreaView style={styles.container}>
      <Text style={styles.title}>Placar</Text>
      <View style={{ marginVertical: 12 }}>
        <Text style={styles.status}>X venceu: {scores.winsX}</Text>
        <Text style={styles.status}>O venceu: {scores.winsO}</Text>
        <Text style={styles.status}>Empates: {scores.ties}</Text>
      </View>

      <TouchableOpacity style={styles.menuButton} onPress={() => setScreen('home')}>
        <Text style={styles.menuButtonText}>Voltar</Text>
      </TouchableOpacity>
    </SafeAreaView>
  );
 }
```

```
  if (screen === 'about') {
    return (
      <SafeAreaView style={styles.container}>
        <Text style={styles.title}>Sobre</Text>
        <Text style={styles.text}>Desenvolvedores:</Text>
        <Text style={styles.text}>Luiz Eduardo Jardim de Medeiros </Text>
        <Text style={styles.text}>Enzo Kurt Sales Almeida </Text>
        <Text style={styles.text}>Tiago Oliveira Castro </Text>
        <Text style={styles.text}>Gustavo Henrique Teixeira Viana </Text>
        <Text style={styles.text}></Text>
        <Text style={styles.text}>Disciplina: React Native</Text>
        <Text style={styles.text}></Text>
        <Text style={styles.text}>Descrição: Este trabalho consiste em um apl
icativo mobile desenvolvido em React Native que implementa o Jogo da Ve
lha. O app possui uma interface com menu principal, tela de jogo, placar pe
rsistente usando AsyncStorage, tela de sobre com informações do desenv
olvedor, e uma SplashScreen exibindo o logo e a versão do aplicativo. O ob
jetivo é demonstrar conhecimentos em React Native, componentes, hooks,
navegação básica e armazenamento local de dados.</Text>
        <Text style={styles.text}></Text>

        <TouchableOpacity style={styles.menuButton} onPress={() => setScre
en('home')}>
          <Text style={styles.menuButtonText}>Voltar</Text>
        </TouchableOpacity>
      </SafeAreaView>
    );
  }

  // tela do jogo
  return (
    <SafeAreaView style={styles.container}>
      <Text style={styles.title}>Jogo da Velha</Text>
      <Text style={styles.status}>{getStatus()}</Text>

      <View style={styles.board}>
        {board.map((value, index) => (
```

```
        <Square key={index} value={value} onPress={() => handlePress(inde
x)} />
       ))}
      </View>

      <View style={{ flexDirection: 'row', marginTop: 20 }}>
       <TouchableOpacity style={[styles.smallButton, { marginRight: 12 }]} on
Press={resetBoard}>
        <Text style={styles.resetButtonText}>Reiniciar</Text>
       </TouchableOpacity>

       <TouchableOpacity style={styles.smallButton} onPress={() => setScre
en('home')}>
        <Text style={styles.resetButtonText}>Menu</Text>
       </TouchableOpacity>
      </View>
    </SafeAreaView>
  );
}

// componente que tenta renderizar a logo, ou um placeholder
function ImagePlaceholder() {
  try {
    // require pode falhar se o arquivo não existir — o try evita crash em tem
po de execução
    const img = require('./assets/logo.png');
    return <Image source={img} style={{ width: 120, height: 120, marginBott
om: 12 }} resizeMode="contain" />;
  } catch {
    return (
      <View style={{
        width: 120, height: 120, borderRadius: 12, backgroundColor: '#eee',
        justifyContent: 'center', alignItems: 'center', marginBottom: 12
      }}>
        <Text style={{ fontWeight: '700', color: '#666' }}>LOGO</Text>
      </View>
    );
  }
```

```
}

const styles = StyleSheet.create({
  container: { flex: 1, alignItems: 'center', justifyContent: 'center', backgroun
dColor: '#fff', padding: 20 },
  title: { fontSize: 28, fontWeight: '700', color: '#222' },
  subtitle: { fontSize: 14, color: '#666', marginBottom: 18 },
  text: { fontSize: 16, color: '#333', marginBottom: 6 },
  menuButton: { backgroundColor: '#007BFF', paddingVertical: 12, padding
Horizontal: 28, borderRadius: 8, marginTop: 12 },
  menuButtonText: { color: '#fff', fontSize: 18, fontWeight: '600' },
  board: { width: 306, height: 306, flexDirection: 'row', flexWrap: 'wrap', bor
derWidth: 2, borderColor: '#333', marginTop: 12 },
  square: { width: '33.33%', height: '33.33%', justifyContent: 'center', alignIt
ems: 'center', borderWidth: 1, borderColor: '#aaa' },
  squareText: { fontSize: 52, fontWeight: '700' },
  playerX: { color: '#E91E63' },
  playerO: { color: '#4CAF50' },
  status: { fontSize: 18, marginTop: 10, color: '#444' },
  resetButtonText: { color: '#fff', fontSize: 16, fontWeight: '700' },
  smallButton: { backgroundColor: '#28A745', paddingVertical: 10, paddingH
orizontal: 18, borderRadius: 8 }
});
```