React Tic-Tac-Toe Game Documentation

04 November 2023 16:43

In this document, we will explore the code and logic behind a Tic-Tac-Toe game developed using React. The game consists of the following components:

App.jsx - The root component that manages game state and handles player moves.
Player.jsx - Component to display and manage player information.
GameBoard.jsx - Component to display and manage the game board.
Log.jsx - Component to log game moves.
GameOver.jsx - Component to display the game result.

App.jsx

The App.js component is the main entry point of the application and handles game state and player moves. Here are some key points about the App component:

useState is used to manage game state, including player names, game turns, and the game board.
The **deriveActivePlayer** function calculates the current active player (X or O) based on the game turns.
The **deriveGameBoard** function constructs the game board based on game turns.
The **deriveWinner** function checks for a winning combination on the game board.
The component handles square selection, game restart, and player name changes.
The Player and **GameBoard** components are rendered, along with a **GameOver** component if there is a winner or a draw.
Player names and turn status are displayed in the UI.

Player.jsx

The Player.jsx component is responsible for displaying and managing player information. It includes player names and an option to change them.

```
import { useState } from 'react';
export default function Player({
  initialName,
  symbol,
  isActive,
  onChangeName,
}) {
  const [playerName, setPlayerName] = useState(initialName);
  const [isEditing, setIsEditing] = useState(false);
  function handleEditClick() {
    setIsEditing((editing) => !editing);
    if (isEditing) {
      onChangeName(symbol, playerName);
    }
  }
  function handleChange(event) {
    setPlayerName(event.target.value);
  }
  let editablePlayerName = <span className="player-name">{playerName}</span>;
  // let btnCaption = 'Edit';
  if (isEditing) {
    editablePlayerName = (
      <input type="text" required value={playerName} onChange={handleChange} />
    );
    // btnCaption = 'Save';
  }
  return (
    <li className={isActive ? 'active' : undefined}>
      <span className="player">
        {editablePlayerName}
        <span className="player-symbol">{symbol}</span>
      </span>
      <button onClick={handleEditClick}>{isEditing ? 'Save' : 'Edit'}</button>
    </li>
  );
}
```

App.jsx

```
import { useState } from 'react';
import Player from './components/Player.jsx';
import GameBoard from './components/GameBoard.jsx';
import Log from './components/Log.jsx';
import GameOver from './components/GameOver.jsx';
import { WINNING_COMBINATIONS } from './winning-combinations.js';
const PLAYERS = {
  X: 'Player 1',
  O: 'Player 2'
};
const INITIAL_GAME_BOARD = [
  [null, null, null],
  [null, null, null],
  [null, null, null],
];
function deriveActivePlayer(gameTurns) {
  let currentPlayer = 'X';
  if (gameTurns.length > 0 && gameTurns[0].player === 'X') {
    currentPlayer = 'O';
  }
  return currentPlayer;
}
function deriveGameBoard(gameTurns) {
  let gameBoard = [...INITIAL_GAME_BOARD.map((array) => [...array])];
  for (const turn of gameTurns) {
    const { square, player } = turn;
    const { row, col } = square;
    gameBoard[row][col] = player;
  }
  return gameBoard;
}
function deriveWinner(gameBoard, players) {
  let winner;
  for (const combination of WINNING_COMBINATIONS) {
    const firstSquareSymbol =
      gameBoard[combination[0].row][combination[0].column];
    const secondSquareSymbol =
      gameBoard[combination[1].row][combination[1].column];
    const thirdSquareSymbol =
      gameBoard[combination[2].row][combination[2].column];
    if (
      firstSquareSymbol &&
      firstSquareSymbol === secondSquareSymbol &&
      firstSquareSymbol === thirdSquareSymbol
    ) {
      winner = players[firstSquareSymbol];
    }
  }
  return winner;
}
function App() {
  const [players, setPlayers] = useState(PLAYERS);
  const [gameTurns, setGameTurns] = useState([]);
  const activePlayer = deriveActivePlayer(gameTurns);
  const gameBoard = deriveGameBoard(gameTurns);
  const winner = deriveWinner(gameBoard, players);
  const hasDraw = gameTurns.length === 9 && !winner;
  function handleSelectSquare(rowIndex, colIndex) {
    setGameTurns((prevTurns) => {
      const currentPlayer = deriveActivePlayer(prevTurns);
      const updatedTurns = [
        { square: { row: rowIndex, col: colIndex }, player: currentPlayer },
        ...prevTurns,
      ];
      return updatedTurns;
    });
  }
  function handleRestart() {
    setGameTurns([]);
  }
  function handlePlayerNameChange(symbol, newName) {
    setPlayers(prevPlayers => {
      return {
        ...prevPlayers,
        [symbol]: newName
      };
    });
  }
  return (
    <main>
      <div id="game-container">
        <ol id="players" className="highlight-player">
          <Player
            initialName={PLAYERS.X}
            symbol="X"
            isActive={activePlayer === 'X'}
            onChangeName={handlePlayerNameChange}
          />
          <Player
            initialName={PLAYERS.O}
            symbol="O"
            isActive={activePlayer === 'O'}
            onChangeName={handlePlayerNameChange}
          />
        </ol>
        {(winner || hasDraw) && (
          <GameOver winner={winner} onRestart={handleRestart} />
        )}
        <GameBoard onSelectSquare={handleSelectSquare}
board={gameBoard} />
```

```
          board={gameBoard} />
        </div>
        <Log turns={gameTurns} />
      </main>
    );
}
export default App;
```

GameBoard.jsx

The GameBoard.jsx component manages the game board, including rendering squares and handling square selection.

```
export default function GameBoard({ onSelectSquare, board }) {
  return (
    <ol id="game-board">
      {board.map((row, rowIndex) => (
        <li key={rowIndex}>
          <ol>
            {row.map((playerSymbol, colIndex) => (
              <li key={colIndex}>
                <button
                  onClick={() => onSelectSquare(rowIndex, colIndex)}
                  disabled={playerSymbol !== null}
                >
                  {playerSymbol}
                </button>
              </li>
            ))}
          </ol>
        </li>
      ))}
    </ol>
  );
}
```

The GameBoard component renders a 3x3 game board with clickable squares. It maps over the board prop to create rows and squares, and the onSelectSquare function is called when a square is clicked. The disabled attribute is used to prevent selecting a square that has already been chosen.

Log.jsx

The Log.jsx component logs the game moves and displays them in the UI.

```
export default function Log({ turns }) {
  return (
    <ol id="log">
      {turns.map((turn) => (
        <li key={`${turn.square.row}${turn.square.col}`}>
          {turn.player} selected {turn.square.row},{turn.square.col}
        </li>
      ))}
    </ol>
  );
}
```

GameOver.jsx

The GameOver.jsx component displays the game result when there is a winner or a draw. It provides an option to restart the game.

```
export default function GameOver({ winner, onRestart }) {
  return (
    <div id="game-over">
      <h2>Game Over!</h2>
      {winner && <p>{winner} won!</p>}
      {!winner && <p>It&apos;s a draw!</p>}
      <p>
        <button onClick={onRestart}>Rematch!</button>
      </p>
    </div>
  );
}
```

The GameOver component displays a message based on whether there is a winner or a draw. It offers a "Rematch" button to restart the game when clicked.

Winning Combinations

The winning-combinations.js file contains the winning combinations for Tic-Tac-Toe. These combinations are used to determine the winner of the game.

```
export const WINNING_COMBINATIONS = [
  [
    { row: 0, column: 0 },
    { row: 0, column: 1 },
    { row: 0, column: 2 },
  ],
  [
    { row: 1, column: 0 },
    { row: 1, column: 1 },
    { row: 1, column: 2 },
  ],
  [
    { row: 2, column: 0 },
    { row: 2, column: 1 },
    { row: 2, column: 2 },
  ],
  [
    { row: 0, column: 0 },
    { row: 1, column: 0 },
    { row: 2, column: 0 },
  ],
  [
    { row: 0, column: 1 },
    { row: 1, column: 1 },
    { row: 2, column: 1 },
  ],
  [
    { row: 0, column: 2 },
    { row: 1, column: 2 },
    { row: 2, column: 2 },
  ],
  [
    { row: 0, column: 0 },
    { row: 1, column: 1 },
    { row: 2, column: 2 },
  ],
  [
    { row: 0, column: 2 },
    { row: 1, column: 1 },
    { row: 2, column: 0 },
  ],
];
```