Añadiendo Redux al Tres en raya

Enrique Barra Arias
Álvaro Alonso González

Repositorio de Github

- Punto de partida: Tic Tac Toe React

  https://github.com/REACT-UPM/ejemplo_tres_en_raya

```
git clone https://github.com/REACT-UPM/ejemplo_tres_en_raya
cd ejemplo_tres_en_raya
npm install
npm start
```

- Ejemplo terminado: Tic Tac Toe Redux

  https://github.com/REACT-UPM/ejemplo_tres_en_raya/tree/redux

Añadir Redux a una app React existente

1. Descargar dependencias

```
npm install react-redux redux
```

1. Definir el estado de la app y ponerlo en src/constants/constants.js
2. Crear el fichero src/redux/actions.js con las acciones
3. Create the file src/redux/reducers.js con los reducers
4. Crear el componente src/redux/ReduxProvider.js
5. Modificar src/index.js para renderizar ReduxProvider en vez de App
6. Modificar src/components/App.js para conectarse con Redux

Paso 2: Definir el estado de la app

src/constants/constants.js

- En apps pequeñas suele ser el mismo que en App.jsx
  - Board: Array
  - Turn: String
  - Moves: Number

- Tenemos que definir el estado inicial de la app (guardaremos esos valore en `app/constants/constants.js`)
  - Tablero vacío
  - Nombre de cada jugador

```
export const PLAYERX = "Player 1 - Xs";
export const PLAYER0 = "Player 2 - 0s";
export const VALUES = [
      ['-', '-', '-'],
      ['-', '-', '-'],
      ['-', '-', '-'],
];
```

Paso 3: Crear fichero `src/redux/actions.js`

- Primera acción **PLAY_POSITION** cada vez que un jugador haga click en una casilla
- ¿Qué información necesita?
  - Qué jugador hizo el movimiento (de quién era el turno)
  - Qué casilla fue clickada

```
export function playPosition (x, y, turn, values) {
    return{
        type: 'PLAY_POSITION',
        x: x,
        y: y,
        turn: turn,
        values: values
    };
}
```

Paso 3: Crear fichero `src/redux/actions.js`

- Segunda acción **RESET** cada vez que un jugador cliquee el botón de reset
- ¿Qué información necesita?
  - Ninguna. Sólo las constantes del estado inicial

```
export function reset() {
    return { type: 'RESET' };
}
```

Paso 4: Crear fichero `src/redux/reducers.js`

- El estado tiene 3 partes (turn, values, moves)
- Usaremos un reducer para cada parte
- Dentro de la carpeta **redux** creamos:
  - `turnReducer.js` (gestiona turn)
  - `gameReducer.js` (gestiona values)
  - `movesReducer.js` (gestiona moves)

- En reducers.js unificaremos/combinaremos las 3 partes para crear el estado global

Paso 4: Crear fichero
src/redux/reducers.js

```
import { combineReducers } from 'redux';
import gameReducer from './gameReducer';
import turnReducer from './turnReducer';
import movesReducer from './movesReducer';

const GlobalState = combineReducers({
    turn: turnReducer,
    values: gameReducer,
    moves: movesReducer,

});


export default GlobalState;
```

Estructura de un reducer

state es la parte del estado que le corresponde a este reducer, NO el estado global

action contiene toda la información necesaria para modificar el estado, específicamente el tipo de acción y el resto de argumentos (payload)

```
// import {..}

function myReducer(state = “DEFAULT_STATE”, action) {
  switch (action.type) {
    case 'ACTION_NAME':
      let newState = Object.assign([], state);
      // … Modify newState
      return newState; // Return the modified state
    default:
      return state;
  }
}
export default myReducer;
```

devuelve el nuevo estado (sólo el trozo correspondiente a este reducer) tras aplicar los cambios necesarios o el estado anterior si nada ha pasado

Paso 5: Crear el resto de reducers

src/redux/turnReducer.js

- Tenemos que cambiar el turno al del otro jugador en el caso de 'PLAY_POSITION' y reiniciar al turno original en el caso de 'RESET'

```
import {PLAYERX, PLAYER0} from '../constants/constants';

function turnReducer(state = PLAYERX, action) {
    switch (action.type) {
    case 'PLAY_POSITION':
        return action.turn === PLAYERX ? PLAYER0 : PLAYERX;
    case 'RESET':
          return PLAYERX;
    default:
        return state;
    }
}

export default turnReducer;
```

Paso 5: Crear el resto de reducers

src/redux/gameReducer.js

- Tenemos que poner una ‘X’ o un ‘0’ en la casilla correspondiente en el caso de 'PLAY_POSITION' y vaciar el tablero en el caso de 'RESET’

```
import { PLAYERX, VALUES } from '../constants/constants';

function gameReducer(state = VALUES, action) {
    switch (action.type) {
    case 'PLAY_POSITION':
        let newValue = action.turn === PLAYERX ? 'X' : '0';
        let newState = JSON.parse(JSON.stringify(state));
        newState[action.x][action.y] = newValue;
        return newState;
    case 'RESET':
          return VALUES;
    default:
        return state;
    }
}
export default gameReducer;
```

Paso 5: Crear el resto de reducers

src/redux/movesReducer.js

- Tenemos que aumentar en 1 el número de movimientos eb 'PLAY_POSITION' y reiniciar a 0 en caso de 'RESET'

```
function movesReducer(state = 0, action) {
    switch (action.type) {
    case 'PLAY_POSITION':
        return state + 1;
    case 'RESET':
        return 0;
    default:
        return state;
    }
}

export default movesReducer;
```

Paso 6: Crear componente

src/redux/ReduxProvider.js

```
import {PLAYERX, VALUES} from '../constants/constants';
import { Provider } from 'react-redux';
import GlobalState from './reducers';
import { createStore } from 'redux';

import React from 'react';
import App from '../components/App';

export default class ReduxProvider extends React.Component {
  constructor(props) {
    super(props);
    this.initialState = { values: VALUES, turn: PLAYERX, moves: 0}
    this.store = createStore(GlobalState, this.initialState);
  }

  render() {
    return (
        <Provider store={ this.store }>
          <div style={{ height: '100%' }} >
            <App store={ this.store } />
          </div>
        </Provider>
    );
  }
}
```

Paso 7: Modificar `src/index.js`

- Ahora el componente raíz es **ReduxProvider** en vez de App.
- Es el que establece la conexión entre la app de React y el DOM real

```
import React from 'react';
import ReactDOM from 'react-dom';
import './assets/styles/index.css';
import ReduxProvider from './redux/ReduxProvider';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
  <React.StrictMode>
    <ReduxProvider />
  </React.StrictMode>,
  document.getElementById('root')
);

reportWebVitals();
```

Paso 8: Modificar
src/components/App.js

- Añadir dependencias:

```
import { connect } from 'react-redux';
import { playPosition, reset } from './../redux/actions';
```

- Cambiar la declaración del componente

```
~~export default~~ class App extends React.Component {
```

- Conectar <App/> con Redux

```
function mapStateToProps(state) {
  return { ...state };
}
export default connect(mapStateToProps)(App);
```

- Ahora tenemos acceso a `this.props.values`, `this.props.turn`, and `this.props.moves` en <App/>
- Para llamar a una acción:

```
this.props.dispatch(playPosition(0,0,PLAYER_X))
this.props.dispatch(reset())
```

Paso 8: Modificar `src/App.js`

- Borrar el estado inicial del constructor (ahora viene de Redux)

```
class App extends React.Component {
  constructor(props) {
    super(props);
    //this.state = {..}
    this.appClick = this.appClick.bind(this);
    this.resetClick = this.resetClick.bind(this);
}
```

- Cambiar el método render para usar las props en lugar del state

```
render() {
   let text = "Turn of " + this.state.turn this.props.turn;
   return (
     <div>
     <Header text={text} />
     <Board values={this.state.values this.props.values}
              appClick={this.appClick}/>
      <Reset resetClick={this.resetClick}/>
    </div>
   );
 }
```

Paso 8: Modificar `src/App.js`

- Modificar el método **appClick**(), ahora toda la lógica va a los reducers

```
appClick(rowNumber, columnNumber) {
    let valuesCopy =
            JSON.parse(JSON.stringify(this.state.values));
    let newMovement = this.state.turn === PLAYERX ? 'X' : '0';
            valuesCopy[rowNumber][columnNumber] = newMovement;
    this.setState({
        turn: this.state.turn === PLAYERX ? PLAYER0 : PLAYERX,
        values: valuesCopy,
        moves: this.state.moves +1 });
    this.props.dispatch(playPosition(rowNumber, columnNumber,
                    this.props.turn));
}
```

Paso 8: Modificar `src/App.js`

- Modificar el método **resetClick**(), ahora toda la lógica va a los reducers

```
resetClick(){
    this.setState({
        turn: PLAYERX,
        values: [
            ['_', '_', '_'],
            ['_', '_', '_'],
            ['_', '_', '_'],
        ],
        winner: undefined,
        moves: 0
    });
    this.props.dispatch(reset())
}
```

Paso 8: Modificar `src/App.js`

```
import React from 'react';
import Header from './Header.jsx';
import Board from './Board.jsx';
import Reset from './Reset.jsx';
import { connect } from 'react-redux';
import { playPosition, reset } from './../redux/actions';

class App extends React.Component {
  constructor(props) {
    super(props);
    this.appClick = this.appClick.bind(this);
    this.resetClick = this.resetClick.bind(this);
    }
    appClick(rowNumber, columnNumber) {
       this.props.dispatch(playPosition(rowNumber, columnNumber, this.props.turn, this.props.values));
    }
    resetClick(){
        this.props.dispatch(reset());
    }
    render() {
            let text = "Turn of " + this.props.turn;

          return (
            <div>
              <Header text={text}/>
              <Board values={this.props.values}  appClick={this.appClick}/>
              <h3>Number of moves: {this.props.moves}</h3>
              <Reset resetClick={this.resetClick}/>
            </div>
          );
      }
}
function mapStateToProps(state) {
  return { ...state };
}
export default connect(mapStateToProps)(App);
```

Cómo añadir nuevas acciones

Cada vez que queremos añadir una acción nueva

1. La definimos en **actions.js:** *type* y *payload*
2. Modificamos los **reducers** para que tengan en cuenta la nueva acción (con un nuevo case statement) si es necesario
3. **Lanzamos** la nueva acción desde App.jsx o desde el componente que sea con **dispatch**