

17. Manipuler des states complexes d'une application React avec Redux

On souhaite créer une application **Tennis Score** qui permet de gérer le score.



1. Créez une nouvelle application React avec cette commande :

```
npx create-react-app react-redux-example
```

2. Ajoutez les packages redux et react-redux en exécutant sur votre shell

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

3. Copier le state, les actions et le reducer dans le fichier [store.js](#).

```
import { createStore } from "redux";  
// state  
const initialState = {  
  player1: 0,  
  player2: 0,  
  advantage: null,  
  winner: null,  
  playing: true,  
};  
// actions creators  
export const playPause = () => ({ type: "playPause" });  
export const restartGame = () => ({ type: "restart" });  
  
export const pointScored = (player) => ({  
  type: "pointScored",  
  payload: { player: player },  
});  
  
function reducer(state = initialState, action) {  
  if (action.type === "restart") {  
    return initialState;  
  }  
}
```



```
if (action.type === "playPause") {
  if (state.winner) {
    return state;
  }
  return {
    ...state,
    playing: !state.playing,
  };
}
if (action.type === "pointScored") {
  const player = action.payload.player;
  const otherPlayer = player === "player1" ? "player2" :
"player1";
  if (state.winner) {
    // On ne peut pas marquer de point si le set est terminé
    return state;
  }
  if (state.playing === false) {
    // On ne peut pas marquer de point si le set est en pause
    return state;
  }
  const currentPlayerScore = state[player];
  if (currentPlayerScore <= 15) {
    // 0 ou 15 => on ajoute 15
    return { ...state, [player]: currentPlayerScore + 15 };
  }
  if (currentPlayerScore === 30) {
    return { ...state, [player]: 40 };
  }
  if (currentPlayerScore === 40) {
    if (state[otherPlayer] !== 40) {
      // Le joueur à gagné
      return { ...state, winner: player };
    }
    if (state.advantage === player) {
      // Le joueur à gagné
      return { ...state, winner: player };
    }
    if (state.advantage === null) {
      // Le joueur a maintenant l'avantage
      return { ...state, advantage: player };
    }
    // L'autre joueur a perdu l'avantage
    return { ...state, advantage: null };
  }
}
return state;
}
export const store = createStore(reducer);
```



4. Copier le CSS dans le fichier index.css.

```
html {  
  font-size: 16px;  
  font-family: "Ubuntu", sans-serif;  
}  
body { margin: 0; }  
* { box-sizing: border-box; }  
#root {  
  padding: 2rem;  
  text-align: center;  
  flex-direction: column;  
  display: flex;  
  align-items: stretch;  
  justify-content: center;  
  min-height: 100vh;  
  max-width: 500px;  
  margin: 0 auto;  
}  
p { margin: 0; }  
.buttons {  
  align-self: center;  
  display: flex;  
  flex-direction: column;  
}  
.buttons-row {  
  display: flex;  
  flex-direction: row;  
  justify-content: space-between;  
}  
.button {  
  font-family: "Ubuntu", sans-serif;  
  margin: 0.5rem;  
  line-height: 1;  
  border: none;  
  padding: 0.8rem 1rem;  
  font-size: 1rem;  
  background: #232536;  
  border-radius: 0.3rem;  
  color: white;  
  font-weight: 300;  
  width: 170px;  
  cursor: pointer;  
}  
.button:hover {  
  background: #2c2f44;  
}
```



```
.button:active {  
  background: #1e212e;  
}  
.display {  
  font-size: 2rem;  
  font-weight: 700;  
  padding: 1rem;  
  background-color: #2759f5;  
  color: white;  
  border-radius: 0.3rem;  
  line-height: 1.5;  
  margin: 0;  
  margin-bottom: 1rem;  
}  
.player-score {  
  font-size: 1.4rem;  
  font-weight: 400;  
  padding: 0.5rem 1rem;  
  margin-bottom: 1rem;  
  background-color: #193ba1;  
  color: white;  
  border-radius: 0.3rem;  
  line-height: 1.5;  
  display: flex;  
  justify-content: space-between;  
}  
.player-games {  
  font-size: 1.4rem;  
  font-weight: 400;  
  padding: 0.5rem 1rem;  
  margin-bottom: 1rem;  
  background-color: #ae2b2b;  
  color: white;  
  border-radius: 0.3rem;  
  line-height: 1.5;  
  display: flex;  
  justify-content: space-between;  
}
```

5. Mettre en place le Provider de React-Redux.

```
<Provider store={store}>  
  <StrictMode>  
    <App />  
  </StrictMode>  
</Provider>,
```



6. Créer des composants pour l'affichage.

```
import { PlayPauseButton } from "../PlayPauseButton";
import { Display } from "../Display";
import { ResetButton } from "../ResetButton";
import { PointScoredButton } from "../PointScoredButton";
export default function App() {
  return (
    <div>
      <Display />
      <div className="buttons-row">
        <PointScoredButton playerId="player1">Point Joueur
1</PointScoredButton>
        <PointScoredButton playerId="player2">Point Joueur
2</PointScoredButton>
      </div>
      <div className="buttons-row">
        <ResetButton />
        <PlayPauseButton />
      </div>
    </div>
  );
}
```

7. Utiliser useSelector et useDispatch pour connecter vos composants à votre store Redux.

Display.js

```
// on import useSelector depuis react-redux
import { useSelector } from "react-redux";

export function Display() {
  // on utilise useSelector avec en paramètre une fonction
  // qui permet de récupérer uniquement la propriété `playing`
  // du state
  const gameIsPlaying = useSelector((state) => state.playing);
  const winner = useSelector((state) => state.winner);
  const player1Score = useSelector((state) => state.player1);
  const player2Score = useSelector((state) => state.player2);
  const advantage = useSelector((state) => state.advantage);

  if (winner) {
    if (winner === "player1") {
      return <p className="display">Joueur 1 gagne</p>;
    } else {
      return <p className="display">Joueur 2 gagne</p>;
    }
  }
}
```

```

    } else if (gameIsPlaying === false) {
      return <p className="display">C'est la pause</p>;
    } else {
      let text = "Le score est: " + player1Score + " - " +
player2Score;
      if (advantage) {
        if (advantage === "player1") {
          text += " avantage joueur 1";
        } else {
          text += " avantage joueur 2";
        }
      }
      return <p className="display">{text}</p>;
    }
  }
}

```

PointScoredButton.js

```

import { useDispatch } from "react-redux";
import { pointScored } from "../store";
export function PointScoredButton({ playerId, children }) {
  const dispatch = useDispatch();
  return (
    <button
      className="button"
      onClick={() => { dispatch(pointScored(playerId)); }}
    > {children}
    </button>
  );
}

```

ResetButton.js

```

import { useDispatch } from "react-redux";
import { restartGame } from "../store";
export function ResetButton() {
  const dispatch = useDispatch();
  return (
    <button
      className="button"
      onClick={() => { dispatch(restartGame()); }}
    > Remettre à zéro
    </button>
  );
}

```

PlayPauseButton.js

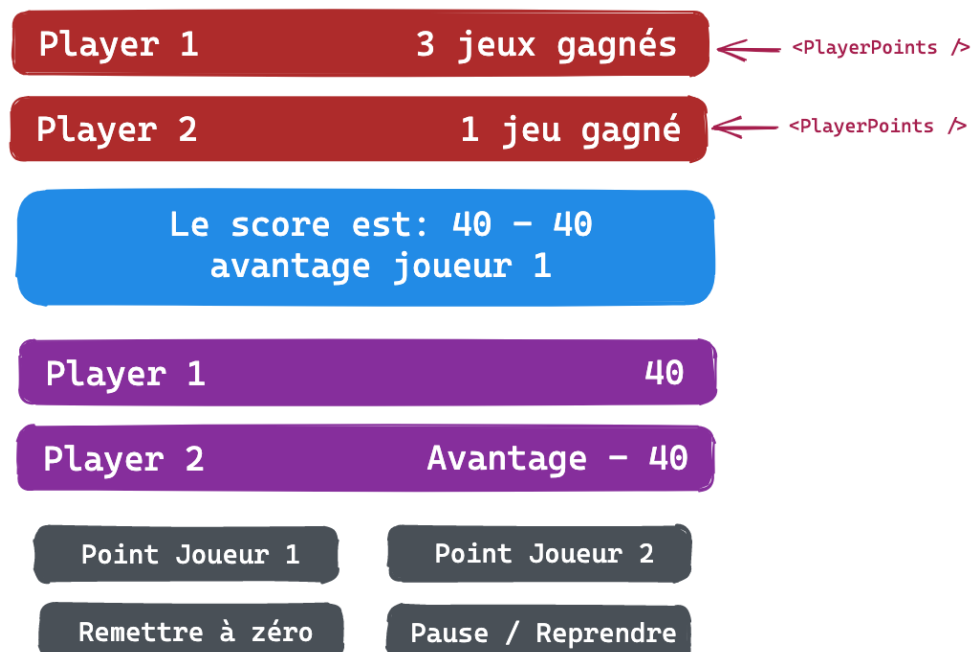
```
import { useDispatch } from "react-redux";
import { playPause } from "../store";
export function PlayPauseButton() {
  const dispatch = useDispatch();
  return (
    <button
      className="button"
      onClick={() => { dispatch(playPause()); }}
    > Pause / Reprendre
    </button>
  );
}
```

8. Mettre en place les deux changements suivants :

- Améliorer la lisibilité du reducer grâce à Immer.
- Ajouter un state pour sauvegarder un historique des jeux de tennis joués.

Astuce : Pour facilement suivre le state de Redux, vous pouvez ajouter le code suivant après la création du reducer (dans le fichier src/store.js) :

9. On souhaite maintenant améliorer l'application afin d'afficher le nombre de match gagné et le score d'un joueur



The mockup shows a tennis game interface. At the top, two red bars represent player scores: 'Player 1' with '3 jeux gagnés' and 'Player 2' with '1 jeu gagné'. Arrows point to these bars with the text '<PlayerPoints />'. Below these is a blue bar showing 'Le score est: 40 - 40' and 'avantage joueur 1'. Underneath are two purple bars for 'Player 1' (score 40) and 'Player 2' (score 'Avantage - 40'). At the bottom are four dark grey buttons: 'Point Joueur 1', 'Point Joueur 2', 'Remettre à zéro', and 'Pause / Reprendre'.

- Créer le composant **PlayerScore** et le connecter à Redux.
- Extraire les selectors dans un fichier selectors.js pour mieux organiser l'application.
- Créer un composant **PlayerPoints** qui affiche le nombre de jeux gagnés par chacun des joueurs (en utilisant la propriété **history** du state).