React State Flow

Review our Tic Tac Toe Game

```tsx
import React, { useState } from 'react'

type Square = 'X' | 'O' | ' '
type Row = [Square, Square, Square]
type Board = [Row, Row, Row]

type Game = {
  board: Board
  id: null | number
  winner: null | string
}
```

```
function App() {
  const [game, setGame] = useState<Game>({
    board: [
      [' ', ' ', ' '],
      [' ', ' ', ' '],
      [' ', ' ', ' '],
    ],
    id: null,
    winner: null,
  })
```

```
async function handleClickCell(row: number, column: number) {
  if (
    // No game id
    game.id === undefined ||
    // A winner exists
    game.winner ||
    // The space isn't blank
    game.board[row][column] !== ' '
  ) {
    return
  }

  // Generate the URL we need
  const url = `https://sdg-tic-tac-toe-api.herokuapp.com/game/${game.id}`

  // Make an object to send as JSON
  const body = { row: row, column: column }
```

```
  // Make a POST request to make a move
  const response = await fetch(url, {
    method: 'POST',
    headers: { 'content-type': 'application/json' },
    body: JSON.stringify(body),
  })

  if (response.ok) {
    console.log('x')
    // Get the response as JSON
    const newGame = (await response.json()) as Game

    // Make that the new state!
    setGame(newGame)
  }
}
```

```
async function handleNewGame() {
  // Make a POST request to ask for a new game
  const response = await fetch(
    'https://sdg-tic-tac-toe-api.herokuapp.com/game',
    {
      method: 'POST',
      headers: { 'content-type': 'application/json' },
    }
  )

  if (response.ok) {
    // Get the response as JSON
    const newGame = (await response.json()) as Game

    // Make that the new state!
    setGame(newGame)
  }
}
```

```
const header = game.winner ? `${game.winner} is the winner` : 'Tic Tac Toe'
```

```
return (
  <div>
    <h1>
      {header} - <button onClick={handleNewGame}>New</button>
    </h1>
    <ul>
      {game.board.map((boardRow, rowIndex) => {
        return boardRow.map((cell, columnIndex) => {
          return (
            <li
              key={columnIndex}
              className={cell === ' ' ? '' : 'taken'}
              onClick={() => handleClickCell(rowIndex, columnIndex)}
            >
              {cell}
            </li>
          )
        })
      })}
    </ul>
  </div>
)
```

Let us extract a component for each cell!

Define the component right in the App.jsx

- Copy/paste the implementation of a specific `li`
- What else is needed?

```
export function Cell() {
  return (
    <li
      className={cell === ' ' ? '' : 'taken'}
      onClick={() => handleClickCell(rowIndex, columnIndex)}
    >
      {cell}
    </li>
  )
}
```

Needed

- rowIndex
- columnIndex
- cell
- something to handle clicking the cell

Wouldn't it be nice to be able to use the parent's state!?

- Well, you cannot...

We can pass down the *parts* of state we need.

- Pass down the cell's value
- Pass down the row
- Pass down the column

```
type CellProps = {
  rowIndex: number
  columnIndex: number
  cell: string
}

export function Cell(props: CellProps) {
  return (
    <li
      className={props.cell === ' ' ? '' : 'taken'}
      onClick={() => handleClickCell(props.rowIndex, props.columnIndex)}
    >
      {props.cell}
    </li>
  )
}
```

Define a local click handler

```
type CellProps = {
  rowIndex: number
  columnIndex: number
  cell: string
}

export function Cell(props: CellProps) {
  function handleClickCell() {
    console.log(`You clicked on ${props.rowIndex} and ${props.columnIndex}`)
  }

  return (
    <li
      className={props.cell === ' ' ? '' : 'taken'}
      onClick={() => handleClickCell(props.rowIndex, props.columnIndex)}
    >
      {props.cell}
    </li>
  )
}
```

This is already better!

```
<ul>
  {game.board.map((boardRow, rowIndex) => {
    return boardRow.map((cell, columnIndex) => {
      return (
        <Cell
          key={columnIndex}
          cell={cell}
          rowIndex={rowIndex}
          columnIndex={columnIndex}
        />
      )
    })
  })}
</ul>
```

But how do we dispatch the API and update the state?

- The `cell` is a read-only `prop` in the `Cell`
- If we did call the API in the cell, how can we *transport* the state to the parent?
- Whatever to do?

State down

- We are sending the state DOWN by doing something like `cell={cell}`
- This *sends* the **PARENT**'s state to the **CHILD** as `props`

Events up

- We still have `handleClickCell` in the parent.
- `handleClickCell` does what we need.
- Rename it to `recordMove`
- And pass **the event handling function** down to the child component

```
<Cell
  key={columnIndex}
  cell={cell}
  rowIndex={rowIndex}
  columnIndex={columnIndex}
  recordMove={recordMove}
/>
```

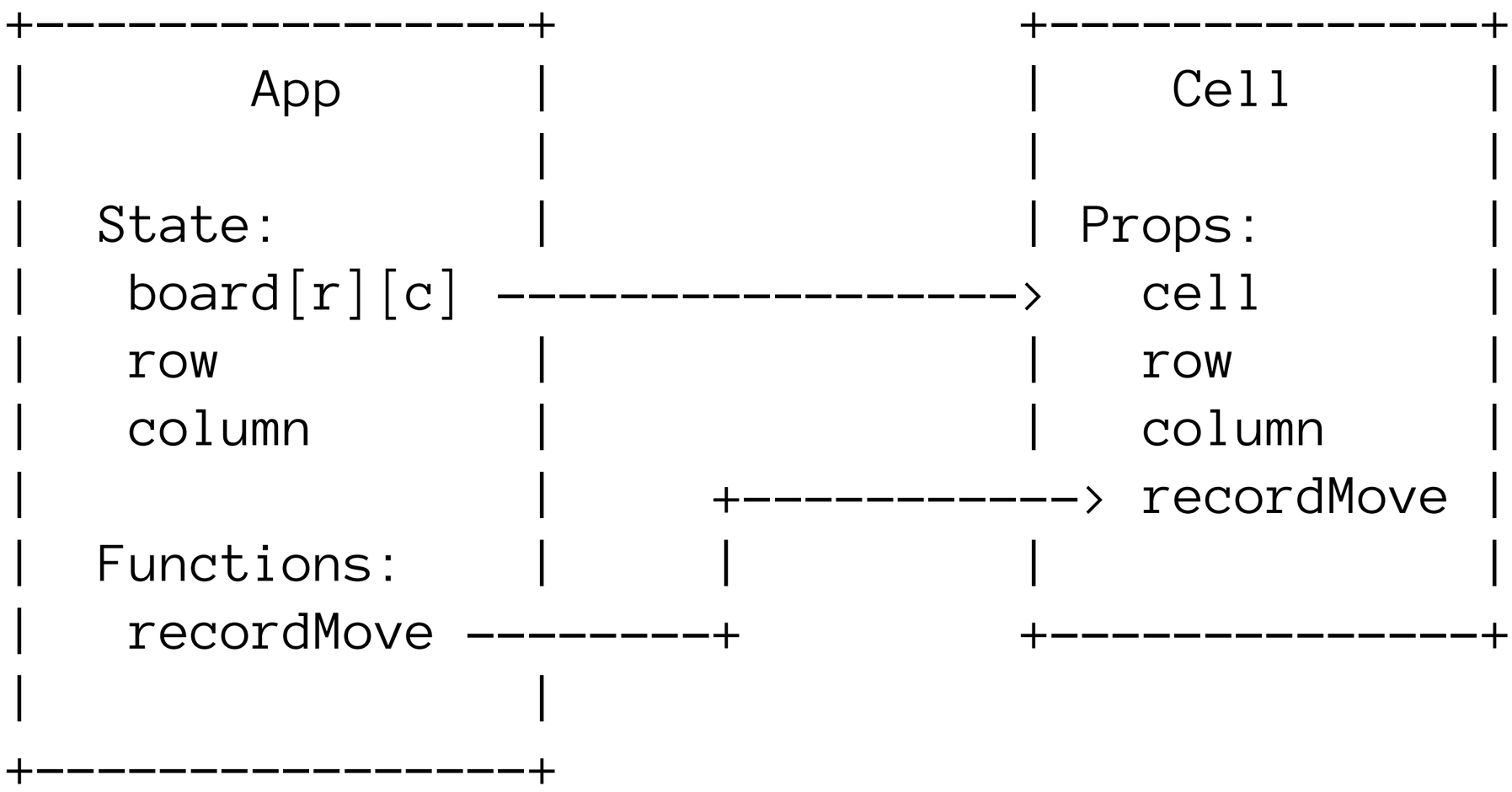
The Cell component can now call UP to the parent's `recordMove`

- This is sending the **event** (something happened) to the parent

```
type CellProps = {
  rowIndex: number
  columnIndex: number
  cell: string
  recordMove: (rowIndex: number, columnIndex: number) => void
}

function handleClickCell() {
  // Send the event UPwards by calling the `recordMove` function we were given
  props.recordMove(props.rowIndex, props.columnIndex)
}
```

Conceptual Model - State Down



Conceptual Model - Events Up

Cell

Cell Receives Click
Calls local onClick

Calls props.recordMove
which is a function.

But the function is *FROM*
the App, so that is the
context of where run

recordMove runs
in the App component

Calls the API
updates state

React sees that state
is updated and re-renders

React makes new `Cell`
components to replace
the old ones

There are new values
for `props.cell`
so the UI draws the
current game.

Code is more DRY

- There is one place for each concept
- The `App`
  - Deals with the API
  - Manages the state
  - Renders the board
  - Tells the Cell where it lives and what to do
- The `Cell`
  - Draws its own UI (an `li`)
  - Knows its row and column
  - Knows its value, and nothing else it needs

Separation of Concerns

- The `Cell` knows only what it needs
- The `App` does not know or care how the `Cell` renders or handles clicks

Missing a Concern?

- Maybe we need a `Game` component?
- Could move the state and the `ul` rendering there.
- Where would the `New Game` button go?
  - Likely move into the `Game` component and user interface

Progress!

Extract that `Cell` component to a new file

Explore some cleanup using TypeScript syntax sugar

- Object shortcut

```
const body = { row: row, column: column }
```

- The key name `row` is the same as the name of the variable holding the value `row`
- Shortcut (structuring the object):

```
const body = { row, column }
```

Flip side.
De-structuring the object

```
props.rowIndex,
props.columnIndex,
props.cell,
props.recordMove,...
```

Destructuring

- If we have an object like

```
const person = {
  name: 'Susan',
  favoriteColor: 'green'
  salary: 1000000
}
```

we can make local variables `name`, `favoriteColor`, and `salary` and initialize their values from the object

```
const { name, favoriteColor, salary } = person
```

```
const { name, favoriteColor, salary } = person
//       ^          ^                ^          v
//       |          |                |          |
//       |          |                |          |
//       ^_________^________________^__________<
```

Notice the { } braces are on the *left*, and the object is on the *right*

Back to our `Cell`

```
type CellProps = {
  rowIndex: number
  columnIndex: number
  cell: string
}

export function Cell(props) {
  function handleClickCell() {
    // Send the event UPwards by calling the `recordMove` function we were given
    props.recordMove(props.rowIndex, props.columnIndex)
  }

  return (
    <li
      className={props.cell === ' ' ? '' : 'taken'}
      onClick={() => handleClickCell(props.rowIndex, props.columnIndex)}
    >
      {props.cell}
    </li>
  )
}
```

Destructuring `props` at the top of a function

```
type CellProps = {
  rowIndex: number
  columnIndex: number
  cell: string
}

export function Cell(props) {
  const { rowIndex, columnIndex, cell, recordMove} = props

  function handleClickCell() {
    // Send the event UPwards by calling the `recordMove` function we were given
    recordMove(rowIndex, columnIndex)
  }

  return (
    <li
      className={cell === ' ' ? '' : 'taken'}
      onClick={() => handleClickCell(rowIndex, columnIndex)}
    >
      {cell}
    </li>
  )
}
```

We can destructure the props right in the function declaration.

```
type CellProps = {
  rowIndex: number
  columnIndex: number
  cell: string
}

export function Cell({ rowIndex, columnIndex, cell, recordMove}) {
  function handleClickCell() {
    // Send the event UPwards by calling the `recordMove` function we were given
    recordMove(rowIndex, columnIndex)
  }

  return (
    <li
      className={cell === ' ' ? '' : 'taken'}
      onClick={() => handleClickCell(rowIndex, columnIndex)}
    >
      {cell}
    </li>
  )
}
```

- ... makes it feel like the properties are nice local variables.
- ... and that syntax is sometimes more straightforward and tidier.

State ↓

Events ↑