Issues With State

One of the biggest issues with state is that any updates to state are made asynchronously. Code that is asynchronous means that it can run while other code is still executing. If it was synchronous, it would wait for other code to finish before it runs.

While this might not be a major issue immediately, it can cause some problems if you're not careful with your design patterns. One common mistake is the following:

1. Handle some form of input / interaction
2. Update state
3. Immediately use the newly updated state in some way

Bearing that in mind, let's take a look at this function handling an input into a field.

```
const [state, updateState] = useState("");

function handleEvent(e) {
  updateState(e.target.value);
  otherFunction(state); // passing what would be the new state
}
```

Because of the asynchronous nature, it is very likely that the argument that would be passed into `otherFunction()` would be the old version of state. However, it's easy to get around this. There are many different methods, but the easiest for something this simple would be this:

```
const [state, updateState] = useState("");

function handleEvent(e) {
  updateState(e.target.value);
  otherFunction(e.target.value); // passing what will be used to update state.
}
```

By doing the above, you're still utilizing state for your component and the React lifecycle, but you're bypassing the asynchronous nature for ONLY that one function call, thereby not having any issues.

TECHNICALLY...

If you want to be super pedantic about it, state updates are technically not asynchronous in the same way that other programming techniques are asynchronous. It's not necesssary to get into the specific details, but knowing precisely how state is updated in React, can sometimes help debug issues. Essentially, changing state just changes it for the **next** render.

If you're interested in how state works in the background, you can look at the AdditionalInfoState.md file.