Session 3 Instructor Guide: Managing Game Flow

Learning Outcomes

By the end of Session 3, students will be able to:

1. **Define state** as data that can change over time and causes components to re-render
2. **Compare state and props** including data flow direction and mutability differences
3. **Explain local state versus shared state** and identify appropriate use cases for each
4. **Define hooks** as functions starting with “use” that provide React features
5. **Identify built-in React hooks** including useState for local state and useContext for shared state
6. **Navigate Context Provider pattern** and explain how custom hooks access shared state
7. **Implement screen navigation** using shared state and conditional rendering
8. **Use constants** for maintainable code and preventing typos
9. **Apply React DevTools** to observe and manipulate state changes in real-time
10. **Create local state** with useState for component-specific data
11. **Access shared state** through custom hooks and Context API
12. **Connect user interactions** to state changes through event handlers

Instruction

Instructor introduces key concepts students need to succeed:

1. **State vs Props Recap** - Review the fundamental difference: props flow down and are read-only, state lives inside components and can change
2. **Local vs Shared State** - Distinguish between component-specific data (local) and app-wide data (shared)
3. **React Hooks Introduction** - Define hooks as React's way to "hook into" features like state and context
4. **Context API Overview** - Explain how Context prevents prop drilling and provides shared state
5. **Constants for Maintainability** - Show how SCREENS constants prevent typos and improve code quality
6. **Conditional Rendering Patterns** - Demonstrate && operator for showing/hiding components based on state
7. **React DevTools Deep Dive** - Use DevTools to visualize state changes and component relationships
8. **Professional State Management** - Introduce patterns for organizing state in larger applications
9. **Game Flow Architecture** - Walk through the screen transition system using state diagrams
10. **Custom Hooks Preview** - Explain how useGame wraps useContext for cleaner component code
11. **Event Handlers and State Updates** - Connect user interactions to state changes through functions
12. **Let's Navigate!** - Overview of today's mission: implement screen navigation and local state

---

Slide Deck Outline

Slide 1: Welcome to State Management! 🧠

- **Title:** “Session 3: Managing Game Flow”
- **Session 2 Recap:** “Last time: Built reusable GameButton with props, styling, and click handlers”
- **Hook:** “Your buttons show alerts — today they’ll actually navigate your game!”
- **Today’s Mission:**
  - **Understand** the difference between state and props
  - **Implement** screen navigation with shared state
  - **Experience** React’s Context API in action
  - **Add** local state for modal functionality
  - **Master** React DevTools for state inspection
- **Visual:** Game screen flow diagram showing SPLASH → PLAYING transition
- **Connection:** “From static components to dynamic, interactive navigation!”

Slide 2: State vs Props - The Data Flow Foundation 📊

- **Title:** “Understanding React’s Data Management”
- **Visual:** Split-screen comparison with arrows showing data flow

Props	State
Flow **down** (parent → child)	Live **inside** components
Read-only (immutable)	**Changeable** (mutable)
Like function parameters	Like component memory
External data	Internal data

- **Analogy:** “Props are like ingredients you receive (can’t change them), State is like your kitchen’s current condition (you control it)”
- **Key Insight:** “Props communicate between components, State manages component behavior”
- **Student Preview:** “You’ll use both today — shared state for navigation, local state for modals”

Slide 3: Local vs Shared State - Choosing the Right Tool 🎯

- **Title:** “When to Use Which Type of State”
- **Visual:** Component tree showing local state bubbles vs shared state umbrella

Local State (useState): - **Scope:** Single component only - **Examples:** Modal visibility, form inputs, toggle states - **Rule:** “If only one component cares, use local state”

Shared State (Context): - **Scope:** Multiple components across the app - **Examples:** User authentication, theme, current screen - **Rule:** “If multiple components need it, use shared state”

- **Today’s Examples:**
  - **Shared:** `screen` state (affects entire app navigation)
  - **Local:** `showCredits` state (only affects SplashScreen modal)
- **Professional Insight:** “Choosing the right state type is a key React skill”

Slide 4: React Hooks - Your State Management Toolkit 🪝

- **Title:** “Hooks: Functions That Hook Into React Features”
- **Definition:** “Functions starting with ‘use’ that provide React capabilities”
- **Key Rules:**
  - Always start with “use” (useState, useContext, useEffect)
  - Only call at the top level of components
  - Can’t be called inside loops or conditions
- **Today’s Hooks:**
  - **useState** - Adds local state to components
  - **useContext** - Accesses shared state from Context
  - **useGame** - Custom hook that wraps useContext for cleaner code
- **Visual:** Hook examples with syntax highlighting
- **Student Connection:** “Hooks are your tools for making components dynamic and interactive”

Slide 5: Context API - Shared State Without Prop Drilling 🌐

- **Title:** “How Context Solves the Prop Drilling Problem”
- **The Problem:** Passing props through multiple component levels
- **Visual:** Before/After diagram showing prop drilling vs Context

Without Context (Prop Drilling):

```
App → SplashScreen → GameButton
  ↓       ↓           ↓
screen  screen      screen
```

With Context:

```
GameProvider (wraps entire app)
    ↓
Any component can access screen directly
```

- **Context Benefits:**
  - **No prop drilling** - Skip intermediate components
  - **Global access** - Any component can access shared data
  - **Clean code** - Less prop passing, more focused components
- **Today's Context:** GameProvider provides screen state to entire app
- **Student Preview:** "Your useGame hook accesses this shared state from anywhere"

Slide 6: Constants - Professional Code Organization 📋

- **Title:** "Why Constants Matter for Maintainable Code"
- **The Problem:** Magic strings scattered throughout code
- **Bad Example:** `if (screen === "splash")` vs `if (screen === "spalsh")` (typo!)
- **Good Example:** `if (screen === SCREENS.SPLASH)` (autocomplete + no typos)

Benefits of Constants: - **Prevent typos** - Autocomplete catches errors - **Single source of truth** - Change once, updates everywhere - **Better refactoring** - IDE can find all usages - **Self-documenting** - Clear intent and available options

- **Today's Constants:** SCREENS object with SPLASH, PLAYING, GAME_OVER
- **Professional Practice:** "Real apps have hundreds of constants for maintainability"

Slide 7: Conditional Rendering - Controlling What Users See 🔀

- **Title:** "Show/Hide Components Based on State"
- **Pattern:** `{condition && <Component />}`
- **How It Works:**
  - If condition is `true` → Component renders
  - If condition is `false` → Nothing renders
- **Examples:**

```
{screen === SCREENS.SPLASH && <SplashScreen />}
{screen === SCREENS.PLAYING && <GameMap />}
{showCredits && <CreditsModal />}
```

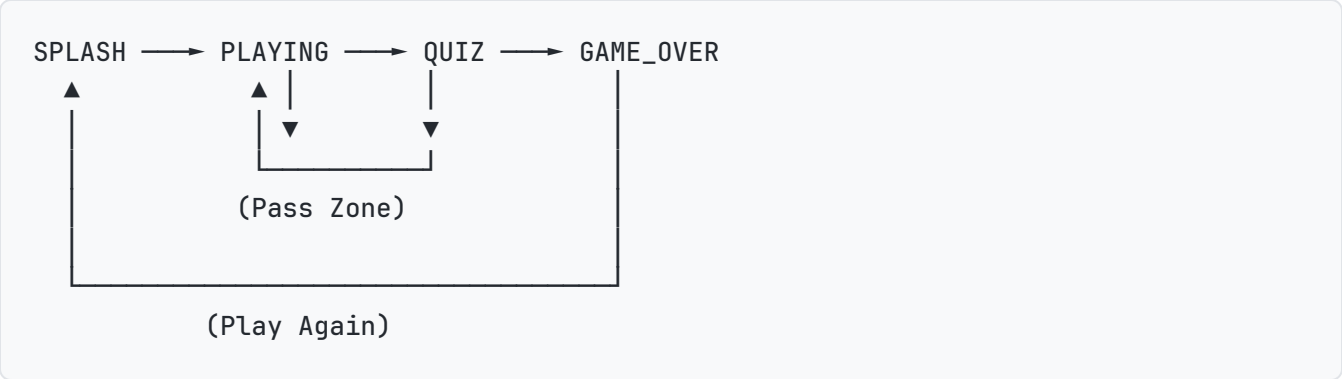
- **Visual:** State diagram showing screen transitions
- **Key Insight:** "One piece of state controls your entire app's display"
- **Student Connection:** "This pattern powers navigation in most React apps"

Slide 8: React DevTools - State Inspector 🔍

- **Title:** "X-Ray Vision for Your App's State"
- **Live Demo:** Show GameProvider in DevTools with state inspection
- **Key Features for State:**
  - **Component tree** - See Provider/Consumer relationships
  - **Hooks section** - View current state values
  - **Real-time updates** - Watch state change as you interact
  - **Manual editing** - Change state values directly for testing
- **Today's Exploration:**
  - Find GameProvider in component tree
  - Inspect screen state value
  - Manually change screen from "splash" to "playing"
  - Watch UI update instantly
- **Professional Usage:** "Essential for debugging state-related issues"

Slide 9: Game Flow Architecture - The Big Picture 🗺️

- **Title:** “How Screen Navigation Works”
- **Visual:** State diagram showing complete game flow



- **Today’s Focus:** SPLASH ↔ PLAYING transition
- **State Control:** Single `screen` variable determines entire app display
- **Future Sessions:** Will add QUIZ and GAME_OVER screens
- **Architecture Insight:** “Complex navigation is just state management”

Slide 10: Custom Hooks - Clean Code Patterns 🎨

- **Title:** “useGame: Wrapping Context for Better Developer Experience”
- **Raw Context Usage:**

```
const context = useContext(GameContext);
const { screen, setScreen } = context;
```

- **Custom Hook Usage:**

```
const { screen, setScreen } = useGame();
```

- **Benefits:**
  - **Cleaner syntax** - Less boilerplate code
  - **Error handling** - Can add validation and error messages
  - **Abstraction** - Hide implementation details
  - **Reusability** - Same hook used everywhere
- **Professional Pattern:** “Custom hooks are how pros organize complex state logic”

Slide 11: Let's Navigate! Today's Coding Mission 🚀

- **Today's Implementation Journey:**
  1. **Explore** SCREENS constants for maintainable navigation
  2. **Add** screen navigation to App.jsx with conditional rendering
  3. **Use** React DevTools to inspect and manipulate state
  4. **Implement** startGame functionality in SplashScreen
  5. **Add** local state for credits modal with useState
  6. **Test** both navigation and modal functionality
- **Success Criteria:**
  - Start Adventure button navigates to GameMap
  - Credits button shows/hides modal
  - React DevTools shows state changes
- **Professional Workflow:** "Build incrementally, test frequently, debug with tools"

[HANDS-ON WORK HAPPENS HERE]

Slide 12: State Management Patterns - Professional Insights 💼

- **Title:** "How Real Apps Organize State"
- **Today's Patterns:**
  - **Context for global state** - App-wide data like current screen
  - **useState for local state** - Component-specific data like modal visibility
  - **Custom hooks for reusability** - Clean interfaces like useGame
  - **Constants for maintainability** - Prevent typos and improve refactoring
- **Scaling Considerations:**
  - **Small apps** - Context + useState (what you're using)
  - **Medium apps** - Add useReducer for complex state logic
  - **Large apps** - External libraries like Redux or Zustand
- **Student Empowerment:** "You're learning patterns used in production apps"

Slide 13: What's Next - Data and APIs 🌐

- **Title:** "Preview of Session 4"
- **Today's Achievement:** "You built navigation with shared and local state"
- **Next Challenge:** "Make your game dynamic with real trivia data"
- **Concepts Coming:**
  - **API integration** - Fetch trivia questions from the internet
  - **Async JavaScript** - Handle network requests and loading states
  - **Data transformation** - Process API responses for your game
  - **Error handling** - Gracefully handle network failures
- **Motivation:** "Your GameMap will show real trivia zones with actual questions!"
- **Visual:** Preview of data-driven game zones