# DAY 5 – Game Logic in React, Conditional Rendering, State Patterns, Grid System & Git/GitHub Basics

---

# 1. Why Games in React?

Games help students understand:

✓ State updates ✓ Conditional rendering ✓ Arrays & indexes ✓ Click events ✓ Turning logic into UI ✓ Thinking like a developer

Tic Tac Toe is the best game to introduce logic building.

---

# 2. Understanding Game Structure (Tic Tac Toe Example)

The board:

- 3×3 grid (9 squares)
- Each square uses:
  - Value (`X` or `O`)
  - Click handler

Basic State Design:

```
const [board, setBoard] = useState(Array(9).fill(null));
const [isXTurn, setIsXTurn] = useState(true);
```

---

# 3. Understanding Grid Layout (CSS)

To create a grid:

```
.board {
  display: grid;
  grid-template-columns: repeat(3, 100px);
  gap: 5px;
}
```

Each cell (square):

```
.square {
  background: white;
  border: 2px solid black;
  font-size: 40px;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100px;
  cursor: pointer;
}
```

---

## 4. Building the Tic Tac Toe Board

Component Structure:

```
App.jsx
 └─ Board.jsx
      └─ Square.jsx
```

---

## 5. Square Component

```
function Square({ value, onClick }) {
  return (
    <button className="square" onClick={onClick}>
      {value}
    </button>
  );
}
```

**Props used:**

- `value` → "X" or "O"
- `onClick` → callback to update board

---

## 6. Board Component

Rendering the grid:

```
<div className="board">
  {board.map((value, index) => (
    <Square
      key={index}
      value={value}
      onClick={() => handleClick(index)}
    />
  ))}
</div>
```

---

## 7. Game Logic: Handling Clicks

```
function handleClick(index) {
  if (board[index] !== null || winner) return;

  const newBoard = [...board];
  newBoard[index] = isXTurn ? "X" : "O";

  setBoard(newBoard);
  setIsXTurn(!isXTurn);
}
```

Explanation:

- Prevent clicking already filled squares
- Update board value
- Switch turns (X → O → X)

---

## 8. Winner Logic (Very Important)

Winning combinations:

```
const winningPatterns = [
  [0,1,2],
  [3,4,5],
  [6,7,8],
  [0,3,6],
  [1,4,7],
  [2,5,8],
  [0,4,8],
  [2,4,6]
];
```

Check winner:

```
function checkWinner(board) {
  for (let [a, b, c] of winningPatterns) {
    if (
      board[a] &&
      board[a] === board[b] &&
      board[a] === board[c]
    ) {
      return board[a];
    }
  }
  return null;
}
```

---

## 9. Calling Winner Function

Inside Board component:

```
const winner = checkWinner(board);
```

---

## 10. Display Winner

```
{winner && <h2>Winner: {winner}</h2>}
```

Or show turn:

```
{!winner && <h2>Turn: {isXTurn ? "X" : "O"}</h2>}
```

---

## 11. Reset Game Button

```
<button onClick={() => {
  setBoard(Array(9).fill(null));
  setIsXTurn(true);
}}>
  Reset Game
</button>
```

---

## 12. Full Tic Tac Toe Code (Combined Example)

```
function TicTacToe() {
  const [board, setBoard] = useState(Array(9).fill(null));
  const [isXTurn, setIsXTurn] = useState(true);

  const winner = checkWinner(board);

  function handleClick(index) {
    if (board[index] || winner) return;

    let newBoard = [...board];
    newBoard[index] = isXTurn ? "X" : "O";

    setBoard(newBoard);
    setIsXTurn(!isXTurn);
  }

  return (
    <div>
      <h1>Tic Tac Toe</h1>

      <div className="board">
        {board.map((value, index) => (
          <Square
            key={index}
            value={value}
            onClick={() => handleClick(index)}
          />
        ))}
      </div>

      {winner && <h2>Winner: {winner}</h2>}

      {!winner && <h2>Turn: {isXTurn ? "X" : "O"}</h2>}

      <button onClick={() => {
        setBoard(Array(9).fill(null));
        setIsXTurn(true);
      }}>
        Reset
      </button>
    </div>
  );
}
```

---

## 13. What Students Learn from Tic Tac Toe

✔ Component communication ✔ Props & callbacks ✔ Array cloning using spread operator ✔ Conditional rendering ✔ Game logic (math + UI) ✔ Grid layout ✔ State flow ✔ Handling events

These concepts help them build:

- Memory games
- Quiz games
- Word scramble
- Minesweeper
- Hangman
- 2048 (advanced)

---

## 14. Git Basics (Must Teach)

Git = Version Control Helps track changes in your project.

---

## 15. Basic Git Commands (Beginner Friendly)

### Initialize Git

```
git init
```

### Check status

```
git status
```

### Add all files

```
git add .
```

### Commit

```
git commit -m "first commit"
```

#### Connect to GitHub

```
git remote add origin <repo-url>
```

#### Push Code

```
git push -u origin main
```

---

## 16. Git Branch Basics

Create branch:

```
git branch feature-ui
```

Switch branch:

```
git checkout feature-ui
```

Merge:

```
git merge feature-ui
```

---

## 17. Why Teach Git on Day 5?

Because now students:

- Know components
- Know state
- Built 3–4 projects
- Are ready to store their code online
- Must learn real developer tools

---

## 18. Mini Tasks for Students

✓ Create a repo for "Tic Tac Toe"

✓ Push code to GitHub

✓ Add README.md

✓ Version control practice

---