Technical Design Document: Flappy Tech Quiz

1. Executive Summary

Project Name: Flappy Tech Quiz **Objective:** Build a "Flappy Bird" style web game where gameplay is interrupted by technical MCQs. Incorrect answers trigger a high-stakes gambling wheel mechanics, and scores are tracked globally. **Target Platform:** Web (Mobile & Desktop). **Timeline:** < 24 Hours (Hackathon Mode). **Deployment:** Vercel.

2. Tech Stack

- **Framework:** Next.js 14+ (App Router).
- **Styling:** Tailwind CSS.
- **Game Rendering:** HTML5 `<canvas>` via React `ref` .
- **Backend/Database: Firebase Firestore** (for real-time Leaderboard).
- **Icons/Assets:** Lucide React, inline SVGs.
- **Deployment:** Vercel.

3. Core Game Mechanics

3.1 The Loop

- **Standard Mechanics:** Gravity, Jump (Space/Tap), Scrolling Pipes.
- **Inputs:** Spacebar/Click (Desktop), Tap (Mobile).

3.2 The Twist: "Stack Overflow" Mode (Quiz Trigger)

- **Trigger:** 5-10% chance upon Jump initiation.
- **Behavior:** Game Pauses -> Quiz Modal Appears -> Timer Starts.

3.3 The "Roulette of Doom" (New Mechanic)

If the user answers a question **INCORRECTLY** (or time runs out), the game does not end immediately. Instead, a **Gambling Wheel** appears.

Wheel Outcomes (Probabilities):

1. **20% - CRITICAL SUCCESS (Second Chance):**
   - The Bird respawns with 1.5 seconds of invincibility.
   - Game resumes.
2. **20% - CRITICAL FAILURE (Instant Death):**
   - Bomb explodes immediately.
   - Game Over screen appears.
3. **60% - SEGFAULT (Jump Failure Status):**
   - Game resumes, but the bird enters "Glitch Mode" for 10 seconds.

- **Effect:** During this mode, every jump input has a **50% chance of being ignored** (simulating packet loss/input lag).
- *Visual:* Bird flickers/glitches transparently.

3.4 Win/Loss States

- **Collision:** Game Over.
- **Wheel (Critical Failure):** Game Over.
- **Wheel (Success/Segfault):** Resume Game.

4. Architecture & Component Structure

4.1 Directory Structure

```
app/
├── layout.tsx
├── page.tsx
├── globals.css
components/
├── GameCanvas.tsx    # Core Physics Loop
├── UIOverlay.tsx     # HUD
├── QuizModal.tsx     # MCQ Interface
├── RouletteWheel.tsx # CSS-based Spinning Wheel
├── Leaderboard.tsx   # Firebase Score List
lib/
├── firebase.ts       # Firebase Config & methods
├── gameLogic.ts      # Collision & Physics
├── questions.ts      # Question Bank
```

4.2 Data Structures

`UserScore` **(Firebase Document)**

```
interface UserScore {
  id?: string;
  username: string; // User entered name
  score: number;    // Pipes cleared
  timestamp: number; // Date.now()
}
```

`Question` **Interface**

```
export interface Question {
  id: string;
  text: string;
  options: string[];
  correctIndex: number;
  category: 'DSA' | 'C++' | 'System Design' | 'React';
}
```

4.3 State Management

Additional State Variables:

- `statusEffect` : `'NONE' | 'GLITCH'` (Active during the 60% failure outcome).
- `showLeaderboard` : `boolean` .
- `showWheel` : `boolean` .

5. Implementation Details

5.1 The Canvas Loop

- **Standard Physics:** `velocity += gravity` .
- **Input Handling with Glitch:**

```
const handleJump = () => {
  if (statusEffect === 'GLITCH') {
    // 50% chance input is ignored
    if (Math.random() > 0.5) return;
  }
  velocity = JUMP_STRENGTH;
};
```

5.2 The Gambling Wheel

- **Component:** `<RouletteWheel />` .
- **Logic:** Simple CSS rotation animation (`transform: rotate(...)`).
- **Result Determination:** Predetermined by JS before the spin starts (`Math.random()`). The wheel visual is just for suspense.
- **Timing:** Spin for 2 seconds -> Pause on result -> Execute Outcome.

5.3 Leaderboard (Firebase)

- **Setup:** Create a Firestore project. Collection name: `flappy-scores` .
- **Write:** At `GAME_OVER` , show input field for "Enter Name". On submit -> `addDoc` .
- **Read:** Fetch top 10 scores ordered by `score` descending.
- **Security:** Allow unauthenticated writes (for Hackathon speed) or use Anonymous Auth.

6. UI/UX Design Specs

- **Wheel UI:**
  - A circular div with 3 slices (Red: Death, Green: Success, Yellow/Glitch: Failure).
  - Overlay text: "FATE DECIDING..."
- **Leaderboard UI:**
  - Tab/Button on Main Menu: "High Scores".
  - Modal pop-up displaying a retro-styled table (Rank | Name | Score).
  - Highlight current user's score if they just played.

7. Step-by-Step Build Plan

1. **Setup:** Next.js + Tailwind setup.
2. **Game Engine:** Build `GameCanvas` (Gravity/Jump/Pipes).
3. **Quiz Module:** Implement the interrupt logic and MCQ Modal.
4. **Wheel Module:**
   - Create visual wheel.
   - Implement logic: `Math.random()` to determine outcome (0-0.2, 0.2-0.4, 0.4-1.0).
   - Connect outcomes to Game State (Resume vs Die).
5. **Glitch Mechanic:** Add conditional logic to the Jump event listener.
6. **Firebase:**
   - Set up Firestore.
   - Create `lib/firebase.ts` .
   - Build `Leaderboard` component.
   - Integrate "Submit Score" form on Game Over screen.
7. **Polish:** Add sound effects (explosion, spin sound), refine assets.
8. **Deploy:** Push to Vercel.

8. Configuration Constants

- `QUIZ_TRIGGER_CHANCE` : 0.08 (8%)
- `WHEEL_PROB_DEATH` : 0.20
- `WHEEL_PROB_RESUME` : 0.20
- `WHEEL_PROB_GLITCH` : 0.60
- `GLITCH_DURATION_MS` : 10000