Super Mario Quiz Quest - Code Review Report

Date: January 14, 2026
Reviewer: GitHub Copilot
Scope: Full codebase review (excluding Gemini API key issues)

🔴 Critical Issues

1. Race Condition in State Updates

Location: App.tsx, App.tsx
Severity: HIGH

Issue: Using `setLives(prev => {...})` inside the game loop creates race conditions. The game loop runs on `requestAnimationFrame`, updating potentially 60 times per second, while React state updates are asynchronous.

```
// Line 318-328 - Enemy collision
setLives(prev => {
  const nextLives = prev - 1;
  if (nextLives <= 0) {
    setGameState(GameState.GAMEOVER);
  } else {
    player.pos = { x: 200, y: 904 };
    player.vel = { x: 0, y: 0 };
    audioService.playIncorrect();
  }
  return nextLives;
});
```

Problems:

- Multiple collisions in the same frame can trigger multiple state updates
- Player state mutation inside state setter is anti-pattern
- No debouncing mechanism to prevent rapid life loss
- Game state change inside life setter violates separation of concerns

Solution: Implement debouncing with timestamp tracking or move collision detection out of the render loop.

---

2. Missing CSS File Reference

Location: index.html
Severity: HIGH

```
<link rel="stylesheet" href="/index.css">
```

Issue: The HTML references `/index.css` but this file doesn't exist in the codebase.

Impact:

- Broken stylesheet link
- Potential console errors
- Missing critical styles

Solution: Either create the file or remove the reference.

---

3. Memory Leak in Game Loop

Location: App.tsx
Severity: HIGH

```
useEffect(() => {
  const handleKeyDown = (e: KeyboardEvent) => { keys.current[e.key] = tru
  const handleKeyUp = (e: KeyboardEvent) => { keys.current[e.key] = false
  window.addEventListener('keydown', handleKeyDown);
  window.addEventListener('keyup', handleKeyUp);
  requestRef.current = requestAnimationFrame(loop);
  return () => {
    window.removeEventListener('keydown', handleKeyDown);
    window.removeEventListener('keyup', handleKeyUp);
    if (requestRef.current) cancelAnimationFrame(requestRef.current);
  };
}, [loop]);
```

Issue: The `loop` function is a dependency that changes every render due to its dependencies (`update` , `draw`). This causes:

- useEffect cleanup and restart on every relevant state change
- Multiple concurrent animation loops
- Event listeners stacking up
- RAF requests accumulating

Solution: Use refs for stable game state or implement proper loop lifecycle management.

---

4. Player State Mutation Outside React

Location: App.tsx
Severity: HIGH

```
const playerRef = useRef({
  pos: { x: 200, y: 904 },
  vel: { x: 0, y: 0 },
  width: 64,
  height: 96,
  grounded: false,
  lastGroundedTime: 0,
  facing: 1 as 1 | -1
});
```

Issue: Critical game state (player position, velocity) is stored in refs and mutated directly, completely bypassing React's state management.

Problems:

- No re-renders triggered when player moves
- Canvas redraws only happen via RAF loop
- Cannot implement React DevTools debugging
- State persistence/serialization impossible
- Difficult to implement features like replay, undo, or save/load

Impact: While this works for performance, it creates a hybrid system that's difficult to maintain and debug.

---

5. Type Safety Violation in Audio Service

Location: services/audioService.ts
Severity: MEDIUM

```
this.ctx = new (window.AudioContext || (window as any).webkitAudioContext
```

Issue: Using `any` type defeats TypeScript's type safety.

Solution:

```
this.ctx = new (window.AudioContext || (window as any as typeof window &
```

Or better, use proper feature detection:

```
const AudioContextClass = window.AudioContext || (window as any).webkitAu
this.ctx = new AudioContextClass();
```

🟠 Major Bugs

6. Incorrect Gemini Model Name

Location: services/geminiService.ts
Severity: MEDIUM

```
model: 'gemini-3-flash-preview',
```

Issue: The model name is incorrect. Gemini 3 doesn't exist (as of this date). The correct models are:

- `gemini-2.0-flash-exp`
- `gemini-1.5-flash`
- `gemini-1.5-pro`

Impact: API calls will fail with a 404 error, forcing fallback to mock data every time.

---

7. Enemy Death Logic Bug

Location: App.tsx
Severity: MEDIUM

```
// Player vs Enemy collision
if (player.vel.y > 0 && player.pos.y + player.height < enemy.pos.y + enem
  enemy.isDead = true;
  player.vel.y = JUMP_STRENGTH * 0.6; // Bounce
  audioService.playStomp();
} else {
  // Hit from side - Enemy also dies when it hits player
  enemy.isDead = true;
  // Reset player and decrement lives
  setLives(prev => {
    // ...
  });
}
```

Issue: The comment says "Enemy also dies when it hits player" but this is not Mario-like behavior. In classic Mario:

- Stomping enemy = player wins
- Side collision = player loses BUT enemy should survive

Current behavior: Enemy dies in both cases, which is incorrect and makes the game too easy.

---

8. Pit Death Recovery Bug

Location: App.tsx
Severity: MEDIUM

```
  if (player.pos.y > CANVAS_HEIGHT) {
    // Pit logic
    setLives(prev => {
      const nextLives = prev - 1;
      if (nextLives <= 0) {
        setGameState(GameState.GAMEOVER);
      } else {
        player.pos = { x: 200, y: 904 };
        player.vel = { x: 0, y: 0 };
        audioService.playIncorrect();
      }
      return nextLives;
    });
    // Temporary safety to prevent infinite loop
    player.pos.y = CANVAS_HEIGHT - player.height - TILE_SIZE;
    player.vel.y = 0;
    player.grounded = true;
  }
```

Issue: The code has contradictory logic:

1. First resets player position to `{ x: 200, y: 904 }` inside state setter
2. Then immediately overrides it to `CANVAS_HEIGHT - player.height - TILE_SIZE`

Result: The "safety" code negates the respawn logic, player appears at bottom of screen instead of spawn point.

---

9. Question Block Collision Detection Flaw

Location: App.tsx
Severity: MEDIUM

```
if (block.type === 'QUESTION' && !block.isHit && gameState === GameState.
  block.isHit = true;
  const currentQ = questions[currentQuestionIndex];
  if (block.label === currentQ.correctAnswer) {
    setFeedback('CORRECT!');
    audioService.playCorrect();
    setTimeout(() => {
      setFeedback(null);
      if (currentQuestionIndex + 1 < questions.length) {
        const nextIndex = currentQuestionIndex + 1;
        setCurrentQuestionIndex(nextIndex);
        initLevel(questions[nextIndex]);
      } else {
        setGameState(GameState.SUCCESS);
        spawnCastleLevel();
      }
    }, 1000);
  }
}
```

Issue: Checking `!feedback` prevents hitting another block while feedback is showing, BUT:

- Wrong answer sets `block.isHit = false` after 800ms
- During that time, `feedback` is still set
- Player cannot try other blocks for 800ms
- This creates confusing UX where player bounces off blocks

Solution: Allow hitting other blocks immediately after wrong answer, only prevent re-hitting the same block.

---

10. Missing Font Loading Strategy

Location: index.html
Severity: LOW-MEDIUM

```
<link href="https://fonts.googleapis.com/css2?family=Press+Start+2P&displ
```

Issues:

- No `font-display` strategy specified (defaults to `swap`)
- No fallback font specified in CSS
- Font loads asynchronously, causing FOUT (Flash of Unstyled Text)
- No preconnect hint to speed up font loading

Solution:

```
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link href="https://fonts.googleapis.com/css2?family=Press+Start+2P&displ
```

And in CSS:

```
font-family: 'Press Start 2P', 'Courier New', monospace;
```

🟡 Code Quality Issues

11. Magic Numbers Throughout

Locations: Multiple files
Severity: MEDIUM

Examples:

```
// App.tsx line 8
const COYOTE_TIME = 150; // ms - Why 150?
const ENEMY_SPEED = 2.0; // Why 2.0?
const INITIAL_LIVES = 3; // Why 3?

// App.tsx line 95
blocks.push({
  // ...
  pos: { x: (centerX - (question.options.length * blockSpacing) / 2) + ic
  // Why 720?
});

// App.tsx line 111
enemies.push({
  // ...
  pos: { x: CANVAS_WIDTH - 300, y: CANVAS_HEIGHT - TILE_SIZE - 64 },
  // Why 300? Why 64?
});

// App.tsx line 364
const pixelSize = 6; // Why 6?
```

Issue: Numerous hardcoded values without explanation or constants.

Solution: Define semantic constants:

```
const ENEMY_SPAWN_OFFSET = 300;
const PLAYER_PIXEL_SIZE = 6;
const QUESTION_BLOCK_Y = 720;
const COYOTE_TIME_MS = 150;
```

---

12. Incomplete Error Handling

Location: services/geminiService.ts
Severity: MEDIUM

```
try {
  const text = response.text;
  if (!text) throw new Error("No response from AI");
  return JSON.parse(text);
} catch (error) {
  console.error("Error fetching questions:", error);
  // Fallback questions if API fails
  return [
    {
      id: 1,
      text: "What is 2 + 2?",
      options: ["3", "4", "5", "6"],
      correctAnswer: "4"
    }
  ];
}
```

Issues:

1. Only returns 1 fallback question but game expects multiple
2. No user notification that AI failed (silently falls back)
3. No retry mechanism
4. No network error differentiation (404 vs network failure vs rate limit)
5. `console.error` in production (should use proper logging)

---

13. Inconsistent Type Definitions

Location: types.ts
Severity: LOW

```
export interface Block extends Entity {
  type: 'QUESTION' | 'FLOOR' | 'CASTLE' | 'BRICK' | 'HIDDEN';
  label?: string;
  isHit?: boolean;
}
```

Issue: The code only uses `'QUESTION'`, `'FLOOR'`, and `'CASTLE'` types. `'BRICK'` and `'HIDDEN'` are defined but never used.

Impact: Dead code, confusing for maintainers.

---

14. No PropTypes or Runtime Validation

Location: components/GameUI.tsx
Severity: LOW

```
interface GameUIProps {
  gameState: GameState;
  currentQuestion?: Question;
```

```
    currentQuestionIndex?: number;
    onStart: (topic: string) => void;
    onStartOffline: () => void;
    feedback: string | null;
    score: number;
    totalQuestions: number;
    lives: number;
  }
```

Issue: `currentQuestion` is optional but used without null checks in JSX:

```
<div className="...">
  {currentQuestion?.text}  // Good - uses optional chaining
</div>
```

But elsewhere in the main component:

```
// App.tsx line 544
currentQuestion={questions[currentQuestionIndex]}
```

If `questions` array is empty, this throws an error.

---

15. Hardcoded Default Topic

Location: components/GameUI.tsx
Severity: LOW

```
const [topic, setTopic] = React.useState('Procure to pay process');
```

Issue: Weirdly specific default topic. Should be:

- Generic (e.g., "Science", "Math", "History")
- Empty string with placeholder
- Or configurable

---

16. Performance: Expensive Inline Calculations in Render

Location: [App.tsx](App.tsx)
Severity: LOW-MEDIUM

```
const drawEnvironment = (ctx: CanvasRenderingContext2D) => {
  // Static background elements
  ctx.fillStyle = 'rgba(255, 255, 255, 0.3)';
  [ {x: 160, y: 100}, {x: 700, y: 200}, {x: 1200, y: 80}, {x: 1600, y: 16
    ctx.beginPath();
    ctx.arc(cloud.x, cloud.y, 30, 0, Math.PI * 2);
    ctx.arc(cloud.x + 30, cloud.y - 16, 50, 0, Math.PI * 2);
    ctx.arc(cloud.x + 60, cloud.y, 30, 0, Math.PI * 2);
    ctx.fill();
  });
  // ... more inline arrays
}
```

Issue: Creating new arrays every frame (60 FPS). These are **static** background elements but recalculated constantly.

Solution: Move to constants:

```
const CLOUD_POSITIONS = [{x: 160, y: 100}, {x: 700, y: 200}, ...];
```

---

17. Canvas Text Rendering Without Measurement

Location: App.tsx
Severity: LOW

```
const words = block.label?.split(' ') || [];
const lines: string[] = [];
let currentLine = "";
words.forEach(word => {
  if ((currentLine + word).length < 15) currentLine += (currentLine ? " "
  else { lines.push(currentLine); currentLine = word; }
});
```

Issue: Text wrapping uses character count (15) instead of actual pixel width measurement.

Problems:

- Doesn't account for font metrics
- "WWW" is much wider than "iii"
- May overflow or waste space

Solution: Use `ctx.measureText()` for accurate wrapping.

---

18. Missing Accessibility Features

Locations: Multiple
Severity: MEDIUM

Issues:

1. No keyboard navigation indicators
2. No ARIA labels
3. No screen reader support
4. No focus management
5. Canvas has no accessible alternative
6. No reduced motion support for animations
7. Color-only feedback (red/green) - not colorblind friendly

Example - Game Over button:

```
<button onClick={() => window.location.reload()}>
  Try Again
</button>
```

Should be:

```
<button
  onClick={() => window.location.reload()}
  aria-label="Restart game"
  autoFocus
>
  Try Again
</button>
```

---

19. No Input Validation

Location: components/GameUI.tsx
Severity: LOW

```
<input
  type="text"
  value={topic}
  onChange={(e) => setTopic(e.target.value)}
  className="..."
/>
<button onClick={() => onStart(topic)}>
  START GAME
</button>
```

Issues:

- No empty string check
- No maximum length validation
- No special character sanitization
- No trim() on whitespace
- Can send empty or malicious strings to AI

Solution:

```
<button
  onClick={() => onStart(topic)}
  disabled={!topic.trim() || topic.length > 200}
>
  START GAME
</button>
```

---

20. Unsafe Key Detection

Location: App.tsx
Severity: LOW

```
const handleKeyDown = (e: KeyboardEvent) => { keys.current[e.key] = true;
const handleKeyUp = (e: KeyboardEvent) => { keys.current[e.key] = false;
```

Issue: Captures **all** keyboard input, including:

- Browser shortcuts (Ctrl+W, Ctrl+T, F5)
- Accessibility shortcuts
- System shortcuts

Solution:

```
  const handleKeyDown = (e: KeyboardEvent) => {
    if (['ArrowUp', 'ArrowDown', 'ArrowLeft', 'ArrowRight', 'w', 'a', 's',
      e.preventDefault();
      keys.current[e.key] = true;
    }
  };
```

🔵 Architecture & Design Issues

21. Tight Coupling: Game Logic in Component

Location: App.tsx
Severity: MEDIUM

Issue: The entire game engine (physics, collision detection, rendering) is inside a React component.

Problems:

- Cannot unit test game logic
- Cannot reuse game engine
- Difficult to optimize
- Hard to add multiplayer or replay features
- React re-renders affect game performance

Recommendation: Separate into layers:

```
/game
  /engine
    - physics.ts
    - collision.ts
    - entities.ts
  /systems
    - renderSystem.ts
    - inputSystem.ts
/components
  - GameCanvas.tsx (thin wrapper)
```

---

22. No State Machine for Game States

Location: App.tsx
Severity: LOW-MEDIUM

Current approach:

```
enum GameState {
  MENU = 'MENU',
  LOADING = 'LOADING',
  PLAYING = 'PLAYING',
  GAMEOVER = 'GAMEOVER',
  SUCCESS = 'SUCCESS'
}
```

Issues:

- No validation of state transitions (can jump from MENU to SUCCESS)
- No state history
- No state-specific cleanup
- Logic scattered across component

Better approach: Use XState or implement a simple FSM with allowed transitions.

---

23. Global Audio Service

Location: services/audioService.ts
Severity: LOW

```
export const audioService = new AudioService();
```

Issue: Global singleton prevents:

- Multiple game instances
- Testing with mocks
- Audio context cleanup
- Settings persistence (mute state)

Better: Use React Context or inject as a dependency.

---

24. No Resource Preloading

Location: Multiple
Severity: LOW

Issue: The game starts immediately without preloading:

- Fonts (Press Start 2P)
- Audio synthesis setup
- Canvas context

This causes:

- First jump sound delay
- Font rendering flash
- Initial jank

Solution: Add a preload phase in LOADING state.

---

25. Hard Refresh on Game Over

Location: components/GameUI.tsx, components/GameUI.tsx
Severity: MEDIUM

```
<button onClick={() => window.location.reload()}>
  Try Again
</button>
```

Issues:

- Loses all state
- Redownloads all assets
- Closes DevTools
- Breaks React navigation
- Poor UX (flash of white, reload delay)

Solution:

```
<button onClick={() => {
  setGameState(GameState.MENU);
  setLives(INITIAL_LIVES);
  setCurrentQuestionIndex(0);
  setFeedback(null);
}}>
  Try Again
</button>
```

🟢 Minor Issues & Code Smells

26. Inconsistent String Formatting

Examples:

```
// Double quotes
import React from "react";
// Single quotes
import { COLORS } from './constants';
// Template literals for static strings
const text = `Generate 5...`;
```

Solution: Pick one style (Prettier recommended).

---

27. Console Logging in Production

Location: services/geminiService.ts
Severity: LOW

```
console.error("Error fetching questions:", error);
```

Issue: Should use proper logging service or environment-based logging.

---

28. No Version Pinning in importmap

Location: index.html
Severity: LOW

```
"react": "https://esm.sh/react@^19.2.3",
```

Issue: `^` allows minor version updates which may break in production.

Solution: Pin exact versions or use lockfile.

---

29. Missing TypeScript Strict Mode

Location: tsconfig.json
Severity: LOW

```
{
  "compilerOptions": {
    "target": "ES2022",
    // Missing strict mode options
  }
}
```

Missing:

```
"strict": true,
"strictNullChecks": true,
"noImplicitAny": true,
"noUnusedLocals": true,
"noUnusedParameters": true,
"noFallthroughCasesInSwitch": true
```

---

30. Redundant requestAnimationFrame Type

Location: App.tsx
Severity: TRIVIAL

```
const requestRef = useRef<number>(undefined);
```

Issue: Type is `number | undefined` but initialized with `undefined`. RAF returns `number`.

Should be:

```
const requestRef = useRef<number | undefined>(undefined);
// Or
const requestRef = useRef<number>(0);
```

📊 Performance Concerns

31. No Canvas Optimization

Location: App.tsx drawing functions
Severity: LOW

Missing optimizations:

- No dirty rectangle tracking (full repaint every frame)
- No off-screen canvas for static elements
- No layer separation (background vs foreground)
- No object pooling for particles/enemies

Impact: Runs fine on desktop but may struggle on mobile.

---

32. Excessive Re-renders

Location: App.tsx
Severity: LOW

Issue: State changes in game loop trigger React re-renders:

```
setFeedback('CORRECT!');  // Triggers render
setLives(prev => prev - 1);  // Triggers render
setGameState(GameState.GAMEOVER);  // Triggers render
```

Each render recalculates `update` and `draw` callbacks due to dependencies.

Solution: Batch state updates or use reducer.

🔒 Security Issues (Non-API Key)

33. No Rate Limiting on AI Calls

Location: components/GameUI.tsx
Severity: MEDIUM

```
<button onClick={() => onStart(topic)}>
  START GAME
</button>
```

Issue: User can spam START GAME button, triggering unlimited AI API calls.

Solution:

```
const [isLoading, setIsLoading] = useState(false);

<button
  onClick={() => onStart(topic)}
  disabled={isLoading || !topic.trim()}
>
  {isLoading ? 'LOADING...' : 'START GAME'}
</button>
```

---

34. Unsafe innerHTML Potential

Location: components/GameUI.tsx
Severity: LOW

```
{currentQuestion?.text}
```

Issue: If AI returns HTML/scripts in question text, React safely escapes it BUT if later changed to `dangerouslySetInnerHTML`, could be XSS vector.

Recommendation: Add content sanitization layer for AI responses.

🧪 Testing Issues

35. Zero Test Coverage

Severity: MEDIUM

Missing:

- No unit tests
- No integration tests
- No E2E tests
- No test configuration (Jest, Vitest, Playwright)

Critical paths to test:

- Collision detection
- State transitions
- API fallback logic
- Score calculation

- Enemy spawning

---

36. Non-Deterministic Game Logic

Location: App.tsx - update function
Severity: LOW

Issue: Game loop uses `performance.now()` and RAF timing, making behavior non-deterministic.

Impact: Cannot write reliable tests or record/replay gameplay.

Solution: Implement fixed timestep or delta time injection for testing.

📱 Responsive & UX Issues

37. No Mobile Support

Severity: MEDIUM

Issues:

- Canvas fixed at 1920x1080
- No touch controls
- Keyboard-only input
- No viewport scaling
- Menu buttons too small for touch

---

38. No Loading Indicators

Location: components/GameUI.tsx
Severity: LOW

```
if (gameState === GameState.LOADING) {
  return (
    <div className="...">
      <div className="text-5xl animate-pulse">GENERATING LEVEL...</div>
    </div>
  );
}
```

Issue: No progress indication. User has no idea if:

- Request is still processing
- API is slow
- Request failed

Better: Add timeout, progress dots, or retry option.

---

39. No Pause Functionality

Severity: LOW

Issue: Cannot pause the game. If user needs to step away:

- Enemies keep moving
- Timer keeps running
- Must take damage or reload

---

40. Score Doesn't Persist

Location: App.tsx
Severity: LOW

```
const [lives, setLives] = useState(INITIAL_LIVES);
```

Issue:

- No high score tracking
- No localStorage persistence
- Cannot compare performances
- No leaderboard support

🐛 Edge Cases & Boundary Conditions

41. Empty Question Array

Location: App.tsx, App.tsx
Severity: MEDIUM

```
if (fetched.length > 0) {
  setCurrentQuestionIndex(0);
  initLevel(fetched[0]);
  setGameState(GameState.PLAYING);
}
```

Issue: If `fetched.length === 0`, game stays in LOADING state forever. No error message or retry.

---

42. Division by Zero Risk

Location: App.tsx
Severity: LOW

```
pos: {
  x: (centerX - (question.options.length * blockSpacing) / 2) + idx * blc
  y: 720
}
```

Issue: If `question.options.length === 0`, blocks spawn at centerX but no guards exist.

---

43. Velocity Can Become NaN

Location: App.tsx
Severity: LOW

```
} else {
  player.vel.x *= 0.8;
}
```

Issue: If `player.vel.x` becomes `NaN` due to any calculation error, it will propagate and break physics.

Solution: Add validation:

```
if (isNaN(player.vel.x)) player.vel.x = 0;
if (isNaN(player.vel.y)) player.vel.y = 0;
```

---

44. Coyote Time Precision Issue

Location: App.tsx
Severity: LOW

```
const canJump = player.grounded || (now - player.lastGroundedTime < COYOT
```

Issue: Uses `performance.now()` which is a high-resolution timestamp but compared against millisecond constant. Works but mixing time sources can cause issues.

---

45. No Maximum Lives Cap

Severity: LOW

Issue: Lives can only decrease, never increase. Fine for current design but if power-ups are added, no upper bound exists.

📐 Code Organization

46. Long Component File

Location: App.tsx - 553 lines
Severity: LOW

Issue: Single file contains:

- Game state
- Physics engine
- Collision detection
- Rendering
- Audio
- Input handling
- UI state
- API calls

Solution: Split into multiple files/hooks:

```
/hooks
  - useGameLoop.ts
  - usePhysics.ts
  - useCollision.ts
  - useInput.ts
/renderers
  - PlayerRenderer.ts
  - EnemyRenderer.ts
  - EnvironmentRenderer.ts
```

---

47. No Barrel Exports

Severity: TRIVIAL

Issue: No `index.ts` files for clean imports. Must import:

```
import { audioService } from './services/audioService';
```

Instead of:

```
import { audioService } from './services';
```

🔧 Configuration & Build Issues

48. Missing .gitignore Entries

Severity: LOW

Likely missing:

- `.DS_Store` (macOS)
- `*.log` (npm logs)
- `.vscode/` (if not shared)
- `coverage/` (test coverage)

---

49. No License File

Severity: LOW

Issue: No LICENSE file. Unclear if code is open source, proprietary, or what rights users have.

---

50. No Package-lock.json / yarn.lock

Severity: MEDIUM

Issue: No lockfile means:

- Non-deterministic builds
- Different developers get different versions
- CI/CD may have different dependencies than local
- `^` ranges in package.json allow breaking changes

📈 Summary Statistics

Category	Count
🔴 Critical Issues	5
🟠 Major Bugs	10
🟡 Code Quality Issues	20
🔵 Architecture Issues	5
🟢 Minor Issues	4
🔒 Security Issues	2
🧪 Testing Issues	2
📱 UX Issues	4
🐛 Edge Cases	5
📐 Organization	2
🔧 Config Issues	3
Total	**62 Issues**

🎯 Priority Recommendations

Must Fix (Before Production):

1. Fix race condition in state updates (#1)
2. Remove missing CSS file reference (#2)

3. Fix memory leak in game loop (#3)
4. Correct Gemini model name (#6)
5. Add rate limiting on API calls (#33)
6. Add package lockfile (#50)

Should Fix (Next Sprint):

7. Fix enemy death logic (#7)
8. Improve error handling (#12)
9. Add input validation (#19)
10. Separate game logic from React (#21)
11. Replace hard refresh with state reset (#25)
12. Add test coverage (#35)

Nice to Have:

13. Add mobile support (#37)
14. Implement pause functionality (#39)
15. Add score persistence (#40)
16. Improve accessibility (#18)
17. Add performance optimizations (#31, #32)

✅ What's Done Well

Despite the issues, these aspects are commendable:

1. ✅ **Clean type definitions** - Good use of TypeScript interfaces
2. ✅ **Separation of concerns** - Services folder structure
3. ✅ **Pixel art implementation** - Creative character rendering
4. ✅ **Audio service** - Nice abstraction for game sounds
5. ✅ **Offline mode** - Good developer experience feature
6. ✅ **Feedback mechanisms** - Visual and audio feedback for actions
7. ✅ **Netlify deployment ready** - Good DevOps setup
8. ✅ **Security checks** - Pre-deploy script is excellent
9. ✅ **Coyote time** - Advanced game feel mechanic
10. ✅ **Responsive controls** - Multiple key bindings

End of Report