Technical Product Requirements Document (PRD)

Project Name: Avatar (MVP Implementation) **Status:** Live / Feature Complete

1. Executive Summary

Vision: "Avatar: Elemento" is a browser-based, turn-based strategy fighting game combining retro 8-bit aesthetics with the elemental combat lore of *Avatar: The Last Airbender*. **Core Problem:** Fans desire a quick, strategic combat experience that faithfully represents the show's "Rock-Paper-Scissors" elemental interactions without requiring complex installation or accounts. **Value Proposition:** A lightweight, responsive web app featuring instant gameplay, unique class mechanics, and nostalgic pixel art visuals using the NES.css framework.

2. User Personas & Stories

Persona A: The Tactical Fan

- **Demographics:** Male/Female, 18-35, Fan of Strategy RPGs.
- **Goal:** Wants to win by exploiting type advantages (e.g., using Water against Fire).

Role	Action	Benefit
Player	Select "Avatar" Class	Access a versatile move set to counter any opponent type.
Player	Use "Stun" or "Evade"	Mitigate incoming damage strategically rather than just attacking.

Persona B: The Casual Browser Gamer

- **Demographics:** Mobile/Desktop users looking for a 5-minute break.
- **Goal:** Immediate fun with zero setup.

Role	Action	Benefit
Player	Open `index.html`	Game loads instantly with no loading screens.
Player	View Battle Log	Clearly understand why an attack dealt extra damage ("Super Effective").

3. Functional Requirements

3.1 Character Roster & Attributes

System: The game must support 5 distinct classes with unique stats and visual assets.

Class	Element	HP	Visual Style	Special Ability
Water Master	Water	100	Blue Robes (Katara style)	**Heal:** Restores 25% Max HP.
Fire Lord	Fire	100	Red Armor (Zuko style)	**Burn:** Applies DOT (Damage Over Time) for 4 turns.
Earth Guard	Earth	120	Green Tunic (Toph style)	**Stun:** Enemy skips next 2 turns.
Air Monk	Air	90	Orange Robes (Tenzin/Gyatso style)	**Evade:** 80% chance to take 0 damage for 4 turns.
The Avatar	Variable	110	Glowing State (Aang style)	**Switch:** Change element/affinity mid-turn.

3.2 Combat Logic (The Core Loop)

Priority: Critical (P0)

1. **Turn Structure:** Player moves -> Logic Calculation -> Animation/Update -> CPU moves.
2. **Move Set:**
    - **Light:** Low Damage (10), 100% Accuracy.
    - **Mid:** Medium Damage (20), 100% Accuracy.
    - **Heavy:** High Damage (30), 100% Accuracy.
    - **Special:** Triggers class-specific effect.

3.3 Power Scaling (Elemental Matrix)

Description: Damage calculation must apply a **1.5x multiplier** based on the attacker vs. defender element.

- **Water** > Fire
- **Earth** > Fire
- **Air** > Earth
- **Fire** > Air
- *(Inverse relationships deal 0.5x damage, neutral deals 1.0x).*

3.4 Game Environment

Priority: High (P1)

- **Map Selection:** Player selects a background which applies a CSS theme/background image to the `.battle-arena` container.
- **Maps:** Air Temple, Fire Nation Ship, Earth Kingdom, Water Tribe.

4. Technical Architecture

4.1 Technology Stack

- **Frontend Framework:** Vanilla JavaScript (ES6+). No build step required.
- **Styling:** CSS3 with **NES.css** (CDN) for 8-bit UI components.
- **Markup:** HTML5.
- **Asset Management:** External URLs for MVP (Images hosted on Itch.io/Pixilart) or local relative paths.

4.2 State Management

The game uses a global state object (`const state`) to track the session.

```
const state = {
    player: { currentHp, maxHp, element, effects: { burn, stun, evade } },
    cpu: { ... },
    turn: 'player' | 'cpu',
    isOver: boolean,
    map: string
};
```

4.3 AI Logic (CPU)

- **Behavior:** Random selection of moves (Light/Mid/Heavy/Special).
- **Timing:** Uses `setTimeout` (1.5s) to simulate "thinking" time before executing a move to improve UX.

5. Data Schema & Relationships

```
classDiagram
    class GameSession {
        +String map
        +Boolean isOver
        +String turn
        +startGame()
```

```
        +endTurn()
    }
    class Character {
        +String name
        +String element
        +Int maxHp
        +Int currentHp
        +Object effects
        +attack()
        +useSpecial()
    }
    class Skill {
        +String type
        +Int damage
        +String effect
    }

    GameSession "1" *-- "2" Character : manages
    Character "1" o-- "4" Skill : possesses
```

6. User Flow & State Logic

```
graph TD
    A[Launch Game] --> B[Character Select Screen]
    B -->|Select Class| C[Map Select Screen]
    C -->|Select Background| D[Battle Initialization]
    D --> E[Player Turn]

    E -->|Select Move| F[Calculate Damage/Effects]
    F --> G{Check HP <= 0}

    G -->|Yes| H[Game Over / Restart]
    G -->|No| I[CPU Turn]

    I -->|AI Action| J[Calculate Damage/Effects]
    J --> K{Check HP <= 0}

    K -->|Yes| H
    K -->|No| E
```

7. UI/UX Design Specifications

- **Typography:** "Press Start 2P" (Google Fonts) for retro readability.
- **Color Palette:**
  - Water: Blue (`#3498db`)
  - Fire: Red (`#e74c3c`)

    - Earth: Green (`#27ae60`)
    - Air: Yellow/Orange (`#f1c40f`)
    - Background: Dark Grey (`#212529`)
- **Feedback:** Battle Log box (`#battle-log`) must update text lines for every action (e.g., "Critical Hit!", "Stunned!").

8. Future Roadmap (Post-MVP)

1. **Multiplayer:** Implement Socket.io for P2P battles.
2. **Sound FX:** Add retro "chiptune" sound effects for attacks and victory.
3. **Local Storage:** Save "Wins/Losses" record in the browser `localStorage`.